PERFORMANCE EVALUATION OF ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING USING 16-BIT IRREGULAR DATA FORMATS

A thesis submitted in partial fulfilment of the requirements for the degree
Of
Bachelor of Technology
In
Electronics and Instrumentation Engineering

BY

Anubhav Mishra
(107EI009)

Swagat Jena
(107EI013)

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled "**Performance Evaluation of Orthogonal Frequency Division Multiplexing using 16-bit Irregular Data Formats**" submitted by **Anubhav Mishra** and **Swagat Jena** in partial fulfilment for the requirements for the award of Bachelor of Technology Degree in Electronics and Instrumentation Engineering at National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Place: Rourkela
Date:

Dr. Sarat Kumar Patra, Ph.D.
Professor
Dept. of Electronics & Communication Engineering
National Institute of Technology
Rourkela – 769008

Abstract

This report asserts that 16-bit Digital Signal Processing applications suffer from dynamic range and noise performance issues. This problem is highly common in complex DSP algorithms and is compounded if they are programmed in high level languages due to no native compiler support for 16-bit data formats. A solution to this problem is achieved by using 16-bit irregular data formats which show significant improvement over fixed and floating point approaches.

First, the data formatting problem for 16-bit programmable devices are defined and discussed. Existing solutions to the problem is taken into consideration. Then a new class of floating point numbers is obtained from which irregular data formats are derived. Attempts are made to derive format with greater dynamic range and noise performance.

Then the irregular data format along with fixed and floating point formats are simulated and analysed for simple DSP applications to make a performance analysis. Finally the data formats under consideration are implemented in a full-fledged Orthogonal Frequency Division Multiplexing model. The inputs and outputs obtained are compared for the percentage of error and final conclusions are drawn. The results indicate that irregular data formats have significant improvement over fixed and floating point formats and 16-bit DSP applications can be implemented in a more effective way using irregular data formats.

Acknowledgements

We are heartily thankful to our supervisor, Professor Sarat Kumar Patra, whose encouragement, guidance and support from the initial to the final level enabled us to develop an understanding of the subject. We owe our deepest gratitude to Manuel Franklin Richey without whose cooperation and guidance the completion of the thesis would not have been possible. He has made his support available by collaborating with us online.

Lastly, we would like to offer our regards and blessings to all of those who supported us in any respect during the completion of the project.

Anubhav Mishra — 107EI009

Swagat Jena — 107EI013

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Introduction

Implementation of complex DSP applications such as Orthogonal Frequency Division Multiplexing is usually done on 32-bit processors for the sake of data integrity and performance. However 32-bit DSP processors are both expensive and slow relative to 16-bit processors due to heavy calculations of higher precision. On the other hand 16 –bit processors suffer from dynamic range and noise performance problems for DSP applications at high speed. The data formats currently available for 16-bit processors are not very effective when it comes to complex DSP applications at high speed. This thesis asserts that actual performance can be improved through the use of irregular data formats.

1.1 Thesis Approach

We approach the 16-bit data formatting problem in the following sequence

1. We understand the hardware implementation of DSP processors and how to program them using High Level Language.

2. We go through data formatting and understand the existing solutions available.

3. We find the advantages and disadvantages of the existing solutions and try to combine their advantages to form a new data format.

4. We perform simulation for an extensive comparison between existing data formats available and the new data format obtained.

5. We summarize the simulation results and reach to conclusions.

1.2 Thesis Organization

The organization of this thesis follows the approach we previously outlined. In chapter 2, we provide an introduction to Digital Signal Processing applications and an overview of DSP processors. In chapter 3, we provide the data formatting problems related to 16-bit processors. In chapter 4, we discuss the existing solutions proposed and available for 16-bit data formatting. We try to obtain their dynamic range and noise performance to make a numerical comparison.

In chapter 5, we move on to derive a new format and discuss its effectiveness based on mathematical calculations. The series of steps involved to derive the new format is discussed in details. Various plots and graph of SNR are obtained to perform a visual comparison.

In chapter 6, we first shortlist the formats to be chosen for comparison and then we move on to simulate the shortlisted formats to make a performance evaluation between them. The simulation results obtained were plotted for an easier analysis

In chapter 7, we finally implement OFDM using all the formats used for comparison. We chose IEEE 802.11a wireless standard for OFDM implementation and test out the shortlisted formats. The results were summarized for a numerical analysis and deciding the best of the lot.

Chapter 2

An Introduction to DSP Applications

An Introduction to DSP applications

Digital Signal Processing is one of the most powerful technologies that have brought revolutionary changes in a broad range of fields. Some of the fields that can be highlighted here are communication, medical imaging, image processing, audio and video processing and the list continues. Working on a core DSP application requires expertise in many fields. Precisely it can be divided into the following sections

- Algorithm development for the specific application
- Language and compiler selection for algorithm coding
- Hardware implementation of the coded algorithm

Development of algorithm is the first and most primary step in the process of implementing a DSP application. The job to be performed by the application is extensively studied. The job is then broken down into a number of modules. Then a series of steps are developed to implement each module. Combining all these modules into one unit gives us the complete algorithm required for implementing the application. Some of the common algorithms used in today's DSP applications are filters, convolution, transforms (such as FFT, DFT etc.)

The second step is to choose a suitable language in order to code the application. Before choosing a language two things must be kept in mind. First is the language's features and capability to implement the application. Second is the compiler's compatibility with hardware available in the market. The compiler should be efficient enough to implement the algorithm with least load on the hardware. As of current standards, most DSP applications are coded using assembly level language and the assembler converts it into the machine code that can be easily implemented in any DSP processor available in the industry.

The third step is to determine a suitable and appropriate hardware that would serve our needs in the most efficient way. The efficiency with respect to cost, implementation, performance and resistance to error (Noise reduction) must be taken into account before selecting hardware. A proper comparison should be made keeping in view the technical specifications that would best suit our DSP application. A wide range of DSP processors are available in the market to choose from. Some of them specialize in image processing, some in encoding/decoding etc. Even for graphics and gaming purposes specialised

Graphics Processing Units (GPUs) are develop that efficiently render high quality video and images.

2.1 An overview of DSP processors

A DSP processor consists of a Multiply/Accumulate Unit (MAC) which performs all its mathematical computations and calculations. This is the core of the processor and its performance and speed solely depends on this unit. Let's take a look at the compact architecture of a DSP processor.

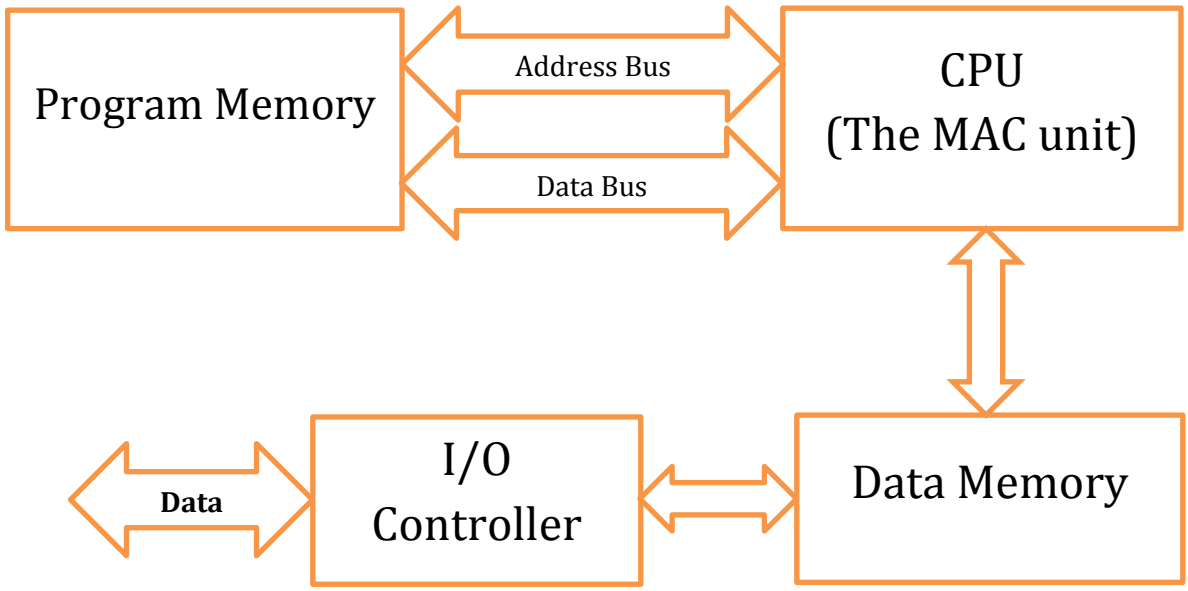

Figure 2-1: Architecture of a DSP Processor

Our discussion mainly deals with the MAC unit. The MAC is broken into three sections, a **multiplier**, an **arithmetic logic unit (ALU) or the accumulator**, and a **barrel shifter**. The multiplier takes the values from two registers, multiplies them, and places the result into another register. The ALU performs addition, subtraction, absolute value, logical operations (AND, OR, XOR, NOT), conversion between fixed and floating point formats, and similar functions. Elementary binary operations are carried out by the barrel shifter, such as shifting, rotating, extracting and depositing segments, and so on.

Chapter 3

Data Formatting in DSP Applications

Data Formatting in DSP applications

The DSP processors come with some standard MAC units. For low end application with less data representation 8-bit DSPs are preferred as they are cheap as well as very fast. For high end applications with large numeric calculations, we have to opt for either a 16-bit processor or a 32-bit one. A 32 bit processor can represent a wide variant of numbers as compared to a 16-bit as the number of register bits are doubled. But this representation comes with cost. Though numbers can be represented more accurately in a 32 bit DSP processor, but the speed of computation slows down as calculations involve very large numbers. Secondly, they come at a pretty steep price in comparison to the 16 bit processor. Let's take a look at the difficulties we face while implementing an algorithm in a 16-bit processor.

Developing software or algorithm for a 16-bit DSP application is pretty difficult in a high level language. Though the high level languages are easier to code, but the data formats available in a standard 16 bit compiler do not provide adequate dynamic range and noise performance. As for example C compiler 16 bit data formats such as int can represent integers from 0 to 65536 in unsigned form or signed integers from -32768 to 32767 in binary format. Any number beyond this leads to an overflow and cannot be represented using the 16 bit int format in C.

The inadequate dynamic range of 16 bit DSP applications operating at high speed forces the application programmer to switch to 32 bit for obtaining a larger dynamic range or to switch to 8 bit for obtaining greater speed as calculations become less complex in the latter case. But again on the flipside 8-bit processors have a very low dynamic range and 32 bit counterparts have a very slow speed and are not suitable for DSP applications at a higher speed.

As a high level language only has standard numeric data formats, it is a difficult job to program a 16-bit DSP application in the high level language. Because after compilation the data formats which the program would use may not be compatible with the hardware specification of a DSP. There is no native format in standard C or C++ that is suitable for signal processing applications. The int format suffers from inadequate dynamic range problem. On the other hand the float type (32 bit) is very comprehensive but when implemented is potentially very slow with C/C++ compiler as well. This is the only reason why many applications are programmed in assembly language.

In spite of all these problems there are compelling reasons why a high level language should be preferred over an assembly language. Firstly programming in a high level language is very easy as compared to assembly level language with not much hassle and pain. Secondly when programmed in assembly language, arithmetic operations can take orders of magnitude longer to execute than equivalent fixed point operations when directly implemented in hardware.

Let's try to find out the shortcomings of data formats available in C family. Native numeric data formats to choose from are –

- Char 8 bits
- Short 16 bits
- Int 16/32 bits (Machine Dependent)
- Long 32 bits
- Float 32 bits
- Double 64 bits

The char format is too small for a decent DSP application as it has only 8 bits. Long and double format execute much slower than int and float. So the normal choice of preference for implement a DSP algorithm is int or float as they execute relatively faster int being the fastest of them all. On a 16-bit processor, the C-int type maps to a 16 bit 2's complement fixed point format and the float type is typically implemented in a 32 bit IEEE standard 754 floating point format.

Neither the int nor the float is optimal for a fundamental 16-bit DSP processor. The reason being that a processor performs a 16 x 16 multiply operation in the multiplier of the MAC unit. The result is a 32 bit word. This word is stored in the accumulator and any further calculation is done with this result in the accumulator itself. The accumulator in a 16 bit processor normally has 32 to 40 bits. After performing all the calculation the result in the accumulator is sent back to the memory by scaling it back to 16 bits.

Issues using the C- data type directly are that when the accumulator sends back the 32 bits data into the 16 bit memory, C truncates the 32 bit value to 16 bit instead of rounding off. This adds up noise in the signal processing application. Another issue is that overflow in C-int is not handled properly. The sum of two very large numbers can result in a binary overflow condition which

yields an erroneous result. As an example 1111 + 11 would yield 0010 in a 4 bit nibble. A better solution to this problem is to cap the result with the largest positive or negative value supported by the format in the case of an overflow. This procedure is called saturation. DSP algorithms can handle saturation with minimal noise but handling overflow is almost impossible. Float in C that is represented as a 32 bit IEEE 754 format doesn't have a native C representation in compiler and has to be hand coded in assembly language due to which its execution takes orders of magnitude higher than the normal int type. Besides the hand coded float type must be provided by the manufacturer as a special library with the compiler. While designing an algorithm in C, explicit calls are to be made to this library to use the float type.

In the world of embedded systems, a jump from a lower (16 bit) to a larger (32 bit) incurs greater system cost and size, but results in a greater system throughput. However that is not the case with a DSP processor. A larger processor will incur a larger cost but may result in a lower throughput. The reason being that the main component in a DSP processor is the Multiply/Accumulate circuit and a 16-bit multiplier can run anytime faster than a 32 x 32 multiplier in a 32 bit processor. So if the algorithm doesn't fit into a 16

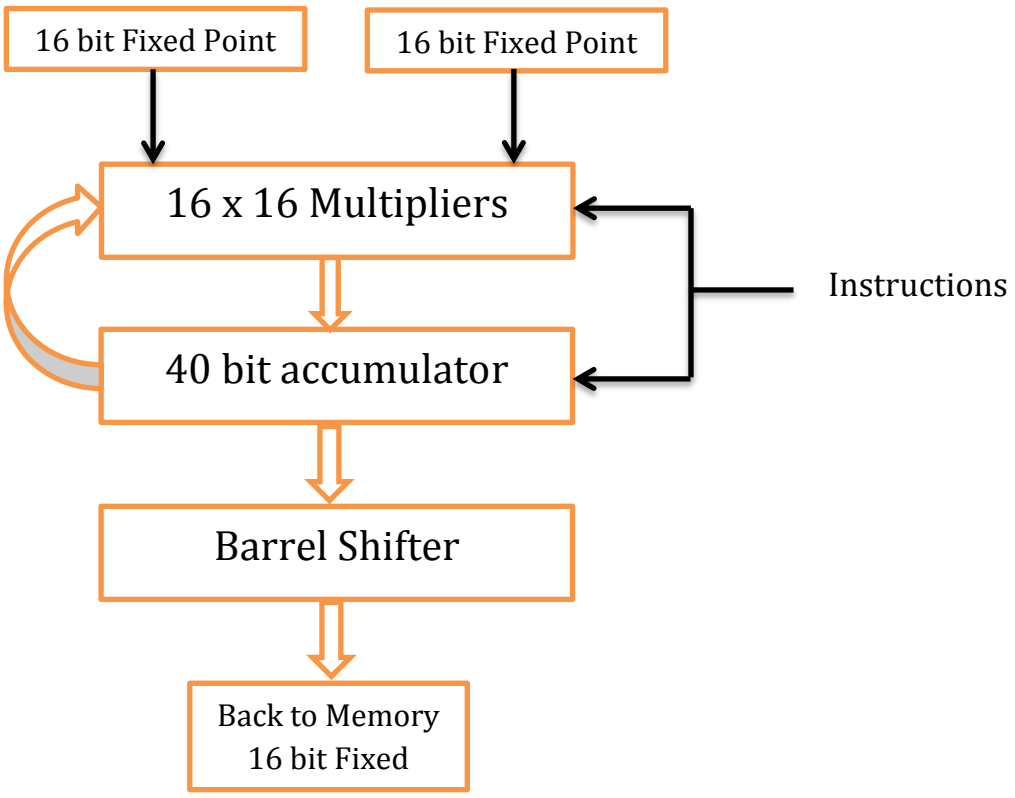

Figure 3-1: MAC Unit in a 16-bit DSP Processor

bit processor due to a limited a limited dynamic range then an unavoidable jump to 32 bit processors are made which results in expensive as well as slow performing processor.

So the two major issues dealt with 16-bit data formatting are –

Dynamic Range - It is defined as the ratio of the largest signal that can be expressed in a data format as compared to the smallest signal.

$$\mathrm{DR} = \frac{Largest\ value\ \ expressed}{Smallest\ value\ expressed}$$

Round-off Noise – Round off noise comes from the error that is created due to rounding the result of an arithmetic operation to the required number of significant digits.

In a DSP application, it is referred to as noise rather than error. For example when a 16 bit fixed point number is multiplied with another 16 bit fixed point number, the result is a 32 bit number. If the result has to be stored back into the same memory than it has to be scaled back to 16 bits which introduces the scaling error or the noise in the signal.

So the main problems addressed and proposed solutions in this thesis are as follows.

- 16 bit DSPs suffer from inadequate dynamic range and noise performance issues due to the use of standard data format. We try to develop a new data format that addresses the two issues and makes a 16 bit DSP comparable to a 32 bit one.
- The problem is still compounded when the DSPs are programmed using a high level language as no new data format that we develop are natively supported by those language compilers. That data format has to be implemented manually by us.

So our aim is to develop an appropriate data format which allows a greater dynamic range and improved noise immunity. Consequently it will be implemented in the C compiler and a performance evaluation will be done with respect to standard data formats available. We try to make an analysis of the new derived format by implementing it in **Orthogonal Frequency Division Multiplexing**.

Chapter 4

Existing Solutions to 16-bit DSP Data Formatting

Existing Solutions to 16-bit DSP Data Formatting

After having a strong hold of the problems faced by the 16-bit DSP applications, we take a look at the existing solutions available and employed widely. There are two approaches that are considered to be the industry standard. One of them is the fixed point approach and the other one is floating point approach. We discuss each of these approaches briefly in terms of their **range** (the largest and smallest numbers they can represent) and their **precision** (the size of the gaps between numbers). For each of these approaches we proceed by calculating their dynamic range and Signal-to-Noise (SNR) ratio.

4.1 Fixed Point Approach

Fixed point representation is used to store *integers*, the positive and negative whole numbers: … -3,-2,-1, 0, 1, 2, 3… The variant of the fixed point approach that are normally used in a DSP application is called fractional fixed point representation. In this representation, each number is treated as a fraction between -1 and 1. The magnitude of each number ranges from 0 to 1. The main advantage of this format is that multiplication doesn't cause overflow, only addition can cause an overflow. Many DSP coefficients and transforms, especially FFT and IFFT, that we will be using in OFDM are typically fractions and can be easily expressed in this format.

To implement this approach, various types of formats are available such as unsigned integer, offset binary, sign and magnitude and two's complement. Of them all, two's complement is the most useful and is normally employed in all the digital systems available. Using 16 bits, two's complement can represent numbers from -32,768 to 32,767. The left most bit is a 0 if the number is positive or zero, and a 1 if the number is negative. Consequently, the left most bit is called the sign bit, just as in sign & magnitude representation. Since each storage bit is a flip-flop, 2's complement is the most convenient and productive format that can be readily implemented with positive as well as negative numbers.

This format was later implemented in the C-compiler resulting in a new version of C called the Embedded C. It handled the fractional format as well as the post multiply accumulator format by introducing two new data types: **fract** and

accum. The fractional format was implemented using the fract data format. On a typical DSP processor, it is a 2's complement 16-bit data word. This format is equivalent to a standard C-int data type multiplied with 2^{-15}. The accumulator is handled by the accum format. On a typical DSP processor, this format is represented as a 32 bit data word with 15 bits above the decimal point to handle addition overflow, 16 bit below the decimal point to represent the fractional part and a single bit for sign representation.

The fract and accum format had the added advantage of rounding off a value while casting it from accum to fract rather than truncating as in native C types. Introduction of these formats improves the noise performance of a 16 bit application but at the end of the day, the 32 bit accum value has to be scaled back to 16 bit fract value. So a noise factor is still introduced by rounding off.

4.2 Floating Point Approach

The floating point number system consists of mantissa and an exponent. For example a decimal floating point number 6.023×10^{23} has the mantissa 6.023 and the exponent 23 with a base 10 as it is in decimal format. This notation is called the scientific notation and is very useful in representing very large and very small numbers. As per this notation the mantissa is normalized so that there is only one non-zero digit to the left of the decimal point. This can be easily achieved by adjusting the value of the exponent. We will be dealing with the binary floating point representation where binary numbers are represented as per the scientific notation discussed above. The difference is that all the operations in a binary floating point format are carried out in base 2 rather than in base 10.

The most common binary floating point formats defined in the IEEE 754 Standard are single precision 32 bit and double precision 64 bit. The single precision 32 bit format is divided into 3 parts

- Bits 0 through 22 form the mantissa (23 bits)
- Bits 23 through 30 form the exponent (8 bits)
- Bit 31 forms the sign bit

These bits form the floating point number in the binary form

$$v = (-1)^S \times M \times 2^{(E-127)}$$

S represents the sign bit. $(-1)^S$ represents the sign. M is the mantissa formed from the 23 bits. Since the mantissa is to be represented in normalized form, only one non-zero digit lies to the left of the binary radix point. As the only non-zero digit in the binary format is 1, it is the only possible digit which will be to the left of the binary point. So it can be considered as an implied bit and needn't be stored in the 23 bits of mantissa which would further increase the precision by another bit.

$$M = 1.m_{22}m_{21}m_{20}\ldots\ldots\ldots\ldots m_1m_0$$

Now coming to the exponent part which has 8 bits, maximum number of values that can be represented is $2^8 = 256$. So in order to represent both positive and negative numbers the distribution can be from -128 to 127. Finally the complete 32 bit single precision can be converted to its equivalent value by

$$v = (-1)^S \times 1.m_{22}m_{21}m_{20}\ldots\ldots\ldots\ldots m_1m_0 \times 2^{(E-127)}$$

Maximum value = $(-1)^S \times 1.1111\ldots\ldots\ldots\ldots 11 \times 2^{(255-127)} = \pm (2 - 2^{-23}) \times 2^{128}$

Minimum value = $(-1)^S \times 1.0000\ldots\ldots\ldots\ldots 00 \times 2^{(0-127)} = \pm 1 \times 2^{-127}$

The format above has been accepted as an IEEE Standard. Now we apply the same floating point approach to the 16 bit formats that can be used in 16 bit applications and calculate the dynamic range as well as the signal to noise ratio for each of those formats and then make a comparison among them to find out which one has a good combination of significant digits (mantissa) and a larger exponent to represent a large range of floating point numbers.

Now let's see how we can divide the 16 bits into different mantissa and exponent bits. One bit has to go for the allotment of the sign bit. We are left with remaining 15 bits. We take a general representation of the form sMeN where

s → sign bit

M → no. of bits for mantissa representation

N → no. of bits for exponent representation

e → separates the exponent and mantissa

A series of representation can be obtained using this format such as s15e0, s14e1, s13e2, s12e3 ……. s0e15. Now we start a comparison of each of these formats on the basis of 2 factors. Let us revisit these factors again:

1) **Dynamic Range** - Ratio of the largest signal that can be expressed in a data format as compared to the smallest signal.
   DR (in dB) = $20 \log_{10} \left[\frac{Largest\ value\ \ expressed}{Smallest\ value\ expressed}\right]$

2) **Peak Signal to peak round off error ratio** – Ratio of the largest mantissa value to the smallest mantissa value considering rounding off of the smallest value. SNR (in dB) = $20 \log_{10} \left[\frac{Largest\ mantissa\ \ expressed}{\frac{1}{2} \times Smallest\ mantissa\ expressed}\right]$

½ is multiplied to consider the round off condition. For example, 0.015 to 0.019 = 0.02, 0.010 to 0.014 = 0.01. Rounding off allows us to accommodate one additional decimal bit of information by introduction of a minimal noise.

We calculate the corresponding values in dB for each of the floating point format and summarize it in a tabular format to make a comparison and analysing there benefits as well as shortcomings.

s15e0 (fractional fixed point format)

Maximum Value = 0.111111111111111 = $\pm (1 - 2^{-15})$

Minimum Value = 0.000000000000001 = $\pm 2^{-15}$

Considering the round-off and increasing one additional bit of representation that can be rounded off we obtain a minimum value = ½ × 2^{-15} = 2^{-16}

Dynamic Range (in dB) = $20 \log_{10} \left(\frac{1 - 2^{-15}}{2^{-16}}\right)$ = 96.3 dB

Peak Signal to Peak Round off Error Ratio = $20 \log_{10} \left(\frac{1 - 2^{-15}}{\frac{1}{2} \times 2^{-16}}\right)$ = 96.3 dB

S14e1

Maximum Value = 1.11111111111111 × $2^{(1-0)}$ = $\pm (2 - 2^{-14}) \times 2^{1}$

Since it is not a standard format we can manipulate it to increase our dynamic range further. So we can represent the implied 1 with an implied 0 to represent further lower numbers though the precision of the number starts decaying after replacing.

Minimum Value = 0.00000000000001 × $2^{(0-0)}$ = $\pm 2^{-14}$

Dynamic Range (in dB) = $20 \log_{10} \left[\frac{(2 - 2^{-14}).2^{1}}{2^{-14}}\right]$ = 96.3 dB

Peak Signal to Peak Round off Error Ratio = $20 \log_{10} \left[\frac{(2 - 2^{-14})}{\frac{1}{2} \times 2^{-14}}\right]$ = 96.3 dB

S13e2

Maximum Value = $1.1111111111111 \times 2^{(3-1)} = \pm (2 - 2^{-13}) \times 2^{2}$

Minimum Value = $0.0000000000001 \times 2^{(0-1)} = \pm 2^{-13} \times 2^{-1}$

Dynamic Range (in dB) = $20 \log_{10} \left[\frac{(2 - 2^{-13}).2^{2}}{2^{-13} \times 2^{-1}}\right]$ = 102.35 dB

Peak Signal to Peak Round off Error Ratio = $20 \log_{10} \left[\frac{(2 - 2^{-13})}{\frac{1}{2} \times 2^{-13}}\right]$ = 90.3 dB

S12e3

Maximum Value = $1.111111111111 \times 2^{(7-3)} = \pm (2 - 2^{-12}) \times 2^{4}$

Minimum Value = $0.000000000001 \times 2^{(0-3)} = \pm 2^{-12} \times 2^{-3}$

Dynamic Range (in dB) = $20 \log_{10} \left[\frac{(2 - 2^{-12}).2^{4}}{2^{-12} \times 2^{-3}}\right]$ = 120.41 dB

Peak Signal to Peak Round off Error Ratio = $20 \log_{10} \left[\frac{(2 - 2^{-12})}{\frac{1}{2} \times 2^{-12}}\right]$ = 84.29 dB

S11e4

Maximum Value = $1.11111111111 \times 2^{(15-7)} = \pm (2 - 2^{-11}) \times 2^{8}$

Minimum Value = $0.00000000001 \times 2^{(0-7)} = \pm 2^{-11} \times 2^{-7}$

Dynamic Range (in dB) = $20 \log_{10} \left[\frac{(2 - 2^{-11}).2^{8}}{2^{-11} \times 2^{-7}}\right]$ = 162.55 dB

Peak Signal to Peak Round off Error Ratio = $20 \log_{10} \left[\frac{(2 - 2^{-11})}{\frac{1}{2} \times 2^{-11}}\right]$ = 78.26 dB

S10e5

Maximum Value = $1.1111111111 \times 2^{(31-15)} = \pm (2 - 2^{-10}) \times 2^{16}$

Minimum Value = $0.0000000001 \times 2^{(0-15)} = \pm 2^{-10} \times 2^{-15}$

Dynamic Range (in dB) = $20 \log_{10} \left[\frac{(2 - 2^{-10}).2^{16}}{2^{-10} \times 2^{-15}}\right]$ = 252.86 dB

Peak Signal to Peak Round off Error Ratio = $20 \log_{10} \left[\frac{(2 - 2^{-10})}{\frac{1}{2} \times 2^{-10}} \right]$ = 72.24 dB

S0e15 (No Mantissa All Exponents)

Maximum Value = $1 \times 2^{(32767\text{-}16383)} = \pm 1 \times 2^{32768}$

Minimum Value = $0.1 \times 2^{(0\text{-}16383)} = \pm 2^{-16383}$

A single decimal point is taken for the round-off consideration which is below the maximum bit represented.

Dynamic Range (in dB) = $20 \log_{10} \left[\frac{1 \times 2^{32768}}{2^{-16384}} \right]$ = 197283.02 dB

Peak Signal to Peak Round off Error Ratio = $20 \log_{10} \left[\frac{1}{2^{-1}} \right]$ = 6.02 dB

Table 4-1: Binary Floating Point Data Formats

Format	Dynamic Range (in dB)	Peak Signal to Peak Round off Error Ratio (in dB)
s15e0	96.3	96.3
s14e1	96.3	96.3
s13e2	102.35	90.3
s12e3	120.41	84.29
s11e4	162.55	78.26
s10e5	252.86	72.24
…	…	…
s0e15	197283.02	6.02

The s10e5 format is the first format that has enough dynamic range to cover both 16-bit integer and 16-bit fractional formats (at least 192 dB). This is an important reason why it has been preferred over other floating point formats. It is clear from the table that dynamic range is improved with 16-bit binary floating point. But as the dynamic range increases more and more noise creeps in and the signal gets weaker. Floating point formats doesn't match fixed point performance unless mantissa (no. of significant bits) is of equal size. Noise performance is often better with fixed point formats than with equivalent sized floating point formats as the SNR ratio for fractional fixed point s15e0 is the maximum. Limitation of the 16-bit floating point formats thus outweighs its advantages. So the question still remains, can we do better with 16-bits?

Chapter 5

Data Formats optimised for 16-bit Signal Processing

Data Formats optimised for 16-bit Signal Processing

We just found that representation of numbers is the most accurate in the fractional fixed point format but trades off some dynamic range. On the other hand, the floating point formats do represent a large range of numbers. But on the downside they introduce way too much error for larger numbers due to which fixed point formats are preferred over them. In this section we try to derive a new data format, which can both represent the signal with less noise compared to floating point formats and which has a larger dynamic range as compared to the fixed point format. Before proceeding to derive new 16-bit data format, the following points must be kept in mind.

- **Word Length** – Must be necessarily 16-bit
- **Efficient Computation** – Must be simple enough to allow multiply accumulate operation
- **Noise Performance** – Must have low round – off noise
- **Dynamic Range** – Must have a large dynamic range
- **Format Mapping** – Format must map to a standard C type for implementation
- **Balanced Range** – Approximately equal dynamic range should be available above and below the decimal point to represent large as well as small numbers

The IEEE 754 floating point 32 bit number system satisfies all the criterions except two. Firstly it is not a 16-bit format and secondly 32 bit numbers are too large for efficient computation in a DSP processor and slows down a multiply accumulate cycle.

Now revisiting the fractional and floating point formats, we find that they have a constant peak signal to peak round off noise ratio up to the point where all the mantissa bits are significant. Once the number of significant bits in the mantissa decreases, the signal to noise ratio rolls off linearly with decreasing significant digits (powers of 2). For example in the s10e5 format, all the 10 mantissa bits remain significant till the implied digit before radix point is 1 i.e. from 1.1111111111 to 1.0000000000. Once the implied 1 changes to an implied 0 to achieve a larger dynamic range, the number of significant digits in mantissa

starts decreasing and correspondingly the SNR ratio i.e. from 0.1111111111 to 0.0000000001 decays. Signal starts decaying as the number of bits decreases from 10 to 1.

Our next step would be to calculate the values of constant as well as decaying SNR for corresponding values of peak signal expressed in fractional format i.e. from 0 to 1. These values are then plotted in a graph and an analysis is made to extract data which helps us in deriving new and effective 16 bit data formats.

s15e0 (fractional fixed point format)

This format has a 0 on the left side of its radix point from the very beginning. There is no implied 1. So signal starts decaying immediately after the top most value. The signal rolls off from its maximum value to its minimum value expressed as a fraction.

Maximum Value = $0.111111111111111 = \pm (1 - 2^{-15}) \approx 1$

Minimum Value = $0.000000000000001 = \pm 2^{-15} = 3.052 \times 10^{-5}$

s14e1

We do not consider this format as we found that it has the same dynamic range and the same value for peak signal to peak round off error value as the s15e0 format.

s13e2

Maximum Value = $1.1111111111111 \times 2^{(3-1)} = \pm (2 - 2^{-13}) \times 2^{2} = 1$ (fractional)

Minimum Value = $0.0000000000001 \times 2^{(0-1)} = \pm 2^{-13} \times 2^{-1}$

Fractional Minimum = $[(2^{-13} \times 2^{-1}) / ((2 - 2^{-13}) \times 2^{2})] = 7.629 \times 10^{-6}$

Decay Point (Point at which signal starts rolling off) = $1.0000000000000 \times 2^{(0-1)} = 2^{-1}$

Fractional Value = $2^{-1} / ((2 - 2^{-13}) \times 2^{2}) = 0.0625$

s12e3

Maximum Value = $1.111111111111 \times 2^{(7-3)} = \pm (2 - 2^{-12}) \times 2^{4} = 1$ (fractional)

Minimum Value = $0.000000000001 \times 2^{(0-3)} = \pm 2^{-12} \times 2^{-3}$

Fractional Minimum = $[(2^{-12} \times 2^{-3}) / ((2 - 2^{-12}) \times 2^{4})] = 9.5379 \times 10^{-7}$

Decay Point (Point at which signal starts rolling off) $=1.000000000000\times2^{(0-3)}=2^{-3}$

Fractional Value = $2^{-3} / ((2 - 2^{-12}) \times 2^{4}) = 3.9067 \times 10^{-3}$

s11e4

Maximum Value = $1.11111111111 \times 2^{(15-7)} = \pm (2 - 2^{-11}) \times 2^{8} = 1$ (fractional)

Minimum Value = $0.00000000001 \times 2^{(0-7)} = \pm 2^{-11} \times 2^{-7}$

Fractional Minimum = $[(2^{-11} \times 2^{-7})/ ((2 - 2^{-11}) \times 2^{8})] = 7.4524 \times 10^{-9}$

Decay Point (Point at which signal starts rolling off) = $1.00000000000 \times 2^{(0-7)}=2^{-7}$

Fractional Value = $2^{-7} / ((2 - 2^{-11}) \times 2^{8}) = 1.52625 \times 10^{-5}$

s10e5

Maximum Value = $1.1111111111 \times 2^{(31-15)} = \pm (2 - 2^{-10}) \times 2^{16} = 1$ (fractional)

Minimum Value = $0.0000000001 \times 2^{(0-15)} = \pm 2^{-10} \times 2^{-15}$

Fractional Minimum = $[(2^{-10} \times 2^{-15})/ ((2 - 2^{-10}) \times 2^{16})] = 2.2748 \times 10^{-13}$

Decay Point (Point at which signal starts rolling off) = $1.0000000000 \times2^{(0-15)}= 2^{-15}$

Fractional Value = $2^{-15} / ((2 - 2^{-10}) \times 2^{16}) = 2.3294 \times 10^{-10}$

Table 5-1: Fractional Binary Floating Point Data Formats

Format	Decay Point	Minimum Value
s15e0	1	3.052×10^{-5}
s13e2	0.0625	7.629×10^{-6}
s12e3	3.9067×10^{-3}	9.5379×10^{-7}
s11e4	1.52625×10^{-5}	7.4524×10^{-9}
s10e5	2.3294×10^{-10}	2.2748×10^{-13}

Based on this result obtained we obtain the following peak signal versus peak round off error plot. It can be easily seen that for values near 1, the fractional fixed point format is the best. However it quickly deteriorates in

performance as soon as the signal value goes below 0.25. s13e2 takes a lead beyond 0.25. Similarly in sequence other floating point formats with lower mantissa overtake the previous ones for smaller signal values.

So in order to get a data format that is better than each of these formats, we need to combine the advantages of each of these formats and recompile a combined format. The **orange line** in the graph shows the signal to noise ratio of the ideal format that combines the best SNR ratios of all the floating point formats. This implies a mantissa with the largest number of significant bits near 1 which gradually decreases as the number gets smaller. The problem with the fixed point fractional format is that significant bits in its mantissa rapidly fall to 0. If we allow the precision to fall at a slower rate, we can simultaneously achieve good noise performance and a wider dynamic range. Now the ideal format represented by the orange line is a mixture of the different floating point formats and hence is random which is very difficult to implement due to its random nature. So we will try to derive formats that can be implemented in an effective manner and would be simultaneously as close to the ideal format as possible.

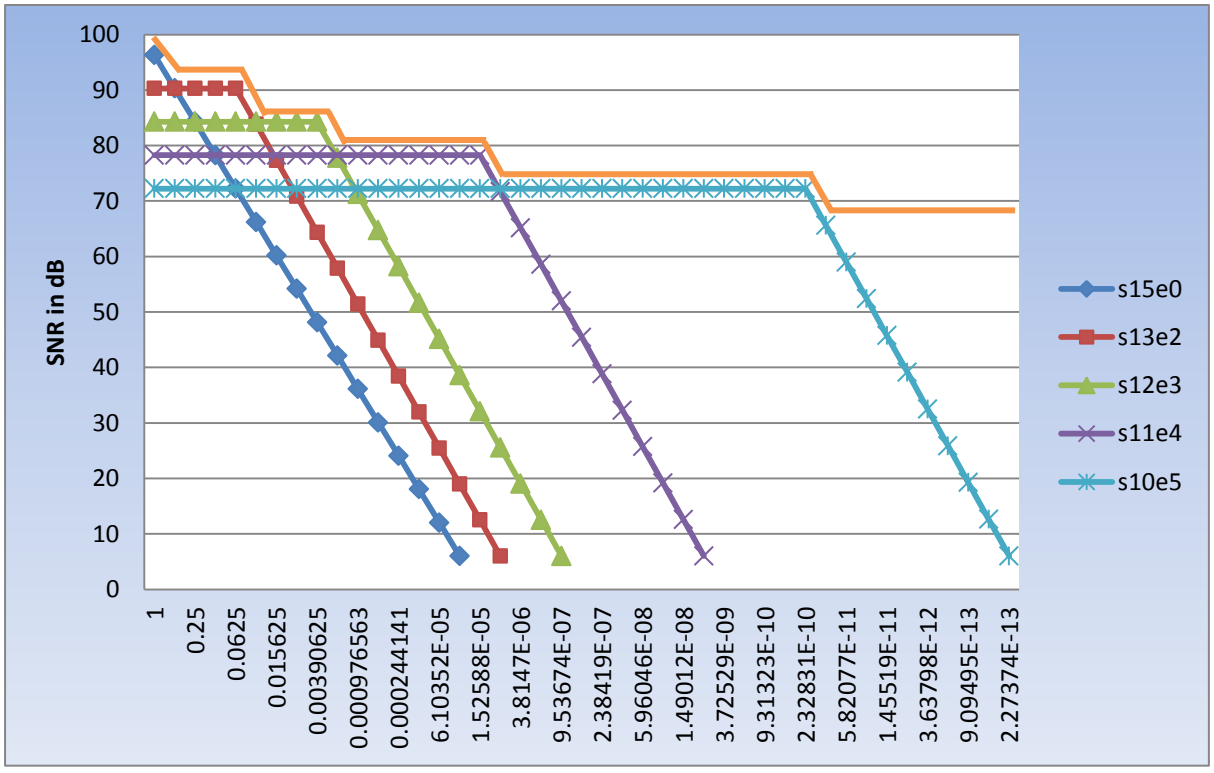

Figure 5-1: Peak Signal Amplitude versus Peak Round off Error

Let us take a look at how the numbers are represented in the fractional fixed point format. The fractional format can be interpreted as a new kind of floating point format by changing the way we look at the number sequence. If we consider all the X's as the mantissa and the mantissa is normalized, then the 1 in the 16 bits can be treated as the implied 1 that lies to the left of the radix point in floating point numbers and the number of leading zeros can represent the exponent E = no. of leading 0s + 1. For example, let us consider the third term in the series S001XXXXXXXXXXXX. Now the mantissa from this can be represented as M = 1. XXXXXXXXXXXX, E = 2(no. of leading zeros) + 1. So the total value will be

$$v = (-1)^S \times 1.\text{XXXXXXXXXXXX} \times 2^{(2+1)}$$

Table 5-2: Fractional Fixed Point Data in Sign-Magnitude Format

Numeric Range	Format S = sign bit X = either 1 or 0	Significant Binary Digits
1.0 – 0.5	S1XXXXXXXXXXXXXX	15
0.5 – 0.25	S01XXXXXXXXXXXXX	14
0.25 – 0.125	S001XXXXXXXXXXXX	13
0.125 – 0.0625	S0001XXXXXXXXXXX	12
0.0625 – 0.03125	S00001XXXXXXXXXX	11
0.03125 – 0.015625	S000001XXXXXXXXX	10
0.015625 – 0.0078125	S0000001XXXXXXXX	9
0.0078125 – 0.00390625	S00000001XXXXXXX	8
0.00390625 – 0.001953125	S000000001XXXXXX	7
0.001953125 – 0.0009765625	S0000000001XXXXX	6
0.0009765625– 0.00048828125	S00000000001XXXX	5
0.00048828125 – 0.000244140625	S000000000001XXX	4
0.000244140625 – 0.0001220703125	S0000000000001XX	3
0.0001220703125 – 0.00006103515625	S00000000000001X	2
0.000030517578125	S000000000000001	1
0	S000000000000000	0

So this fractional format can be re-casted as a floating point format

$$v = (-1)^S \times 1.M \times 2^E$$

where 1 can be treated both as an implied 1 as well as a separator between mantissa and exponent. Now let us represent this same format with trailing zeros and replacing the X's with M's to specify the mantissa. Exponent is determined by the number of trailing zeros i.e. E = n+1. The separator 1 as in previous case becomes an implied 1.

Table 5-3: Fractional Fixed Point Data re-casted with trailing zeros

Numeric Range	Format S = sign bit M = mantissa 0 = exponent 1 = separator	Significant Binary Digits
1.0 – 0.5	SMMMMMMMMMMMMMM1	15
0.5 – 0.25	SMMMMMMMMMMMMM10	14
0.25 – 0.125	SMMMMMMMMMMMM100	13
0.125 – 0.0625	SMMMMMMMMMMM1000	12
0.0625 – 0.03125	SMMMMMMMMMM10000	11
0.03125 – 0.015625	SMMMMMMMMM100000	10
0.015625 – 0.0078125	SMMMMMMMM1000000	9
0.0078125 – 0.00390625	SMMMMMMM10000000	8
0.00390625 – 0.001953125	SMMMMMM100000000	7
0.001953125 – 0.0009765625	SMMMMM1000000000	6
0.0009765625– 0.00048828125	SMMMM10000000000	5
0.00048828125 – 0.000244140625	SMMM100000000000	4
0.000244140625 – 0.0001220703125	SMM1000000000000	3
0.0001220703125 – 0.00006103515625	SM10000000000000	2
0.000030517578125	S100000000000000	1
0	S000000000000000	0

Until this point, we haven't achieved anything new as we are just viewing the same fixed point fractional format only with a different perspective. One thing that should be noted here is that, changing our point of view we obtain a format which has variable number of bits allotted to mantissa and exponents as opposed to standard fixed and floating point number systems. Now we have to devise out methods how this factor can benefit us for our purpose.

Applying the leading and trailing zeros mechanism and dividing the number of zeros on either side of the mantissa separated with a 1 on each side, we can obtain multiple formats for a single combination of mantissa and exponents. Out of the 15 combinations, we can obtain a total of 120 representations which can be included in a new class. This class can now act as parent for derivation of specific data formats.

Table 5-4: New Class of Floating Point Formats

Format **S = sign bit** **M = mantissa** **0 = exponent**	**Significant Binary Digits**	**Identifier (for each set of M and E)**
S1MMMMMMMMMMMMM1	14	A1
S1MMMMMMMMMMMM10, S01MMMMMMMMMMMM1	13	B1 B2
S1MMMMMMMMMMM100, S01MMMMMMMMMMM10, S001MMMMMMMMMM1	12	C1 C2 C3
S1MMMMMMMMMM1000, S01MMMMMMMMMM100, S001MMMMMMMMM10, S0001MMMMMMMMM1	11	D1 D2 D3 D4
5 values	10	E1-E5
……	……	……
S110000000000000, . . . S000000000000011	1	N1-N14
S100000000000000, . . . S000000000000001	1	O1-O15
S000000000000000	0. Note: S=1 could be reserved for irregular data such as NAN or ∞, but S=0 should represent 0.	P1, P2

Some unique features of this class are given as follows

- **Variable Precision** – Variable combination of mantissa and exponents to give different precision levels for different number ranges.
- **Mantissa Combination** – Mantissa from each of the category with similar exponent patterns can be all combined to give a mantissa of higher pattern.
- **Dual Exponent Mapping** – A number's actual value can be represented by $v = (-1)^S \times 1.M \times 2^{f(EL,ER)}$. EL represents the exponent function involving number of zeros to the left and ER represents the exponent function involving number of zeros to the right. Exponent is represented as a combined function of ER and EL. So the ways in which exponent can be represented increases with two functions as opposed to one comes into play.

Before we start deriving new data formats from this class, we need to keep the following points in mind.

1. The format must have a path into C data types i.e. it must be capable of being represented by int formats.
2. It should have adequate dynamic range.
3. The mantissa should roll off gradually allowing greater precision for larger numbers near to 1.

We make a first attempt to develop a fractional format which contains each and every term in the class along with appropriate number of implied zeros to represent the number ranges associated to them. As can be seen in the table, the dynamic range of this format is huge as we have 120 representations for different number ranges between 0 and 1. But the major problem with this format is that the significant bits in the mantissa rolls off from 14 to 13 at the range 0.5 to 0.25. The signals represented by the fractions 0.25 to 0.5 are pretty large and larger number of significant bits in the mantissa would be definitely preferred.

Table 5-5: A First Attempt at the Fractional Format

Numeric Range	Format S = sign bit , M = Mantissa, 0 = Exponent, 1 = Separator, Red = Actual Digit, Black = Implied Digit	Significant Binary Digits	Identifier (from Class)
1.0 – 0.5	S0.1MMMMMMMMMMMMM1	14	A1
0.5 – 0.25	S0.01MMMMMMMMMMMM1	13	B2
0.25 – 0.125	S0.001MMMMMMMMMMMM10	13	B1
0.125 – 0.0625	S0.0001MMMMMMMMMMM10	12	C2
0.0625 – 0.03125	S0.00001MMMMMMMMMMM100	12	C1
0.03125 – 0.015625	S0.000001MMMMMMMMMMM1	12	C3
0.015625 – 0.0078125	S0.0000001MMMMMMMMMM1	11	D4
0.0078125 – 0.00390625	S0.00000001MMMMMMMMMM1000	11	D1
0.00390625 – 0.001953125	S0.000000001MMMMMMMMMM100	11	D2
0.001953125 – 0.0009765625	S0.0000000001MMMMMMMMMM10	11	D3
…….	…….	…	…
	S0. “105 0s” 000000000000001	1	O15
0.0	0000000000000000	0	P1

Now our primary objective is to increase the number of significant digits at higher values of signal. We can make a second attempt to achieve greater precision at the top and prevent the roll off of mantissa quickly at the top by parting with some amount of dynamic range. The combining mantissa property of the class allows us to get a value with a higher precision. To increase a precision bit at the top, what we can do is combine one instance of from each class with similar formatting. In our case we try to combine the mantissa of all those instances which have a common pattern at the right.

Table 5-6: A Second Attempt at the Fractional Format (New Format)

Numeric Range	Format S = sign bit , M = Mantissa, 0 = Exponent, 1 = Separator, Red = Actual Digit, Black = Implied Digit	Significant Binary Digits	Identifier (from Class)
1.0 – 0.5	S0.1MMMMMMMMMMMMM1	14	A1
0.5 – 0.25	S0.01MMMMMMMMMMMMM10	14	B1, C2, D3,…, O14
0.25 – 0.125	S0.001MMMMMMMMMMMM1	13	B2
0.125 – 0.0625	S0.0001MMMMMMMMMMMM100	13	C1, D2, …
0.0625 – 0.03125	S0.00001MMMMMMMMMMM1	12	C3
0.03125 – 0.015625	S0.000001MMMMMMMMMMM1000	12	D1, E2, …
0.015625 – 0.0078125	S0.0000001MMMMMMMMMM1	11	D4
0.0078125 – 0.00390625	S0.00000001MMMMMMMMMM10000	11	E1, F2, …
0.00390625 – 0.001953125	S0.000000001MMMMMMMMM1	10	E5
…….	…….	…	…
	S0. “14 0s” 000000000000001	1	O15
0.0	0000000000000000	0	P1

For instance, let us consider the instances with a trailing pattern of 10.

S1MMMMMMMMMMMM10	B1
S01MMMMMMMMMMM10	C2
S001MMMMMMMMMM10	
S0001MMMMMMMMM10	
……	…
S000000000000010	O14

Now we add the mantissa to obtain one additional bit of precision adding a 1 preceding to it for separator and appending the trailing pattern 10 at the end.

The only exponent field in this case would be the number of zeros in the trailing pattern.

S MMMMMMMMMMMM	13 bits
S MMMMMMMMMMM	12 bits
S MMMMMMMMMM	11 bits
S MMMMMMMMM	10 bits
S MMMMMMMM	9 bits
.....	...
S M	1 bit
SMMMMMMMMMMMMM	14 bits

So the final mantissa increases by 1 bit and hence adds to the precision. The other counterpart with equivalent 14 significant bits is A1. So this would lead to a format with either leading 0s or trailing 0s i.e. with a single exponent field and an increase in bit precision at the top. We name this format as the **New Format**.

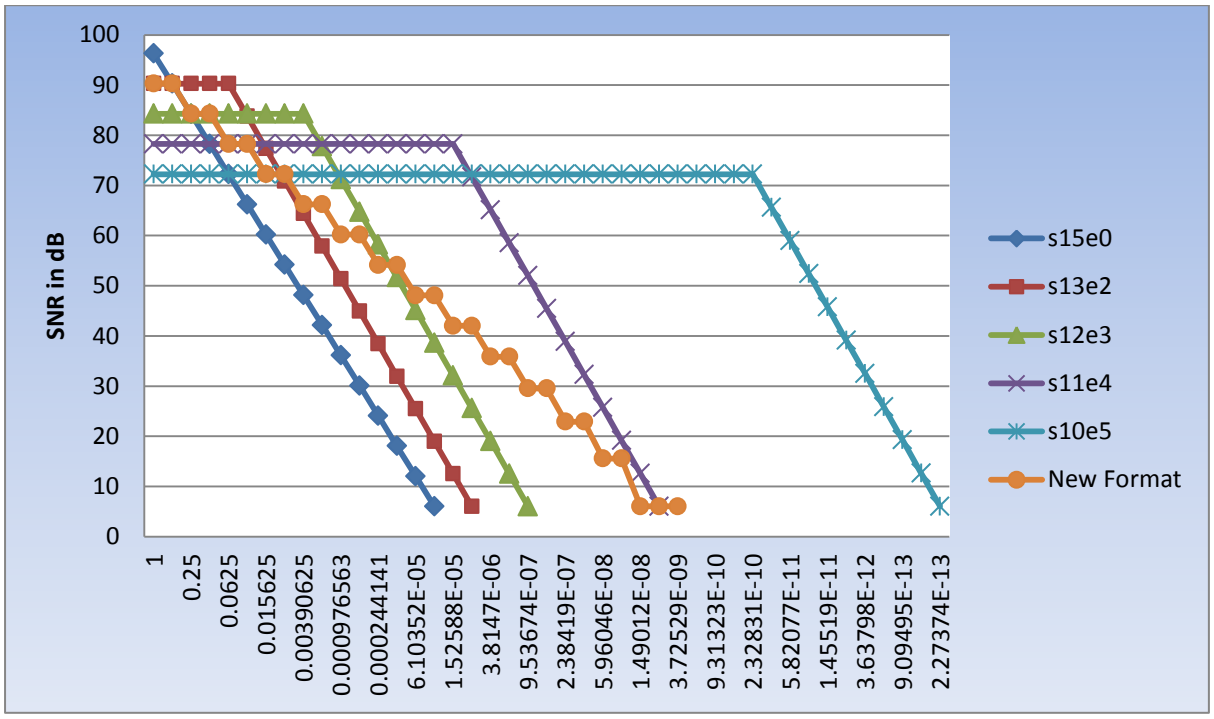

Figure 5-2: New Format comparison with fixed and floating point formats

As can be seen from the graph, this format lags behind fraction format only between the range 1 to 0.5. For rest of the range, this format takes a lead and is better than rest of the floating point formats in either the dynamic range or the SNR ratio. It gives us an optimum balance between the two and serves our purpose.

A function was devised to calculate the Peak Signal Amplitude versus Peak Round off error ratio for the 'New Format' whose plot is obtained as shown in the figure.

Code

```
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
	float SNR[31];
	int m = 13,i=1;
	long double ratio;
	const float base = 2;
	for (int i = 1; i<=29; i=i+2)
	{
			ratio = (2-pow(base,-m))/(pow(base,-m)* 0.5);
			SNR[i] = SNR[i+1] = 20 * log10(ratio);
			m--;
	}
	for (int j = 1; i<=30; i=i++)
	{
		std::cout<<SNR[i]<<"\n";

	}

	printf("Hit any key to terminate program\n");
	getchar();
	return 0;
}
```

Output

90.3085 90.3085 84.2873 84.2873 78.2657 78.2657 72.243 72.243 66.2181 66.2181 60.189 60.189 54.1514 54.1514 48.0967 48.0967 42.0074 42.0074 35.8478 35.8478 29.5424 29.5424 22.9226 22.9226 15.563 15.563 6.0206 6.0206 6.0206

Figure 5-3: Function for SNR calculation of New Format

Now we focus on the implementation part i.e. how the bits can be decoded to obtain the number we want to represent it with. The values can be implemented in a C program by the following interpretation of the bits.

- For values with last bit $m_0 = 1$ i.e. A1, B2, C3, D4…..

We detect the first 1 while traversing from MSB to LSB (excluding the sign bit) i.e. while moving from m_{14} to m_0 and count the number of 0s encountered in our path. We then select our mantissa by removing the first 1 that we encounter and the LSB which is also a 1 and adding a radix point and an implied one before it. For example, let us consider the case of C3. It's represented in the New Format as S0.00001MMMMMMMMMMM1 which covers a range of 0.0625 – 0.03125. Now the 16-bits represented in memory are S001MMMMMMMMMM1.

Count of leading zeros n = 2

Mantissa = 1. MMMMMMMMMM

Exponent can be given by the function = 2n + 1

So value v = $(-1)^S \times 1.\text{MMMMMMMMMM} \times 2^{-(2n+1)}$

We see that the expression for v satisfies our requirements.

- For values with last bit $m_0 = 0$ i.e. those obtained from mantissa combination

We detect the first 1 while traversing from LSB to MSB i.e. while moving from m_0 to m_{14} and count the number of 0s encountered in our path. We then select our mantissa by removing the first 1 that we encounter and adding a radix point and an implied one before m_{14}. For example, let us consider the case of 6th term S0.000001MMMMMMMMMM1000 which covers a range of 0.03125 – 0.015625. Now the 16-bits represented in memory are SMMMMMMMMMM1000.

Count of trailing zeros n = 3

Mantissa = 1. MMMMMMMMMM

Exponent can be given by the function = 2n

So value v = $(-1)^S \times 1.\text{MMMMMMMMMM} \times 2^{-2n}$

So the complete representation of this New Format is

$$\text{Value } v = \begin{cases} (-1)^S \times 1.\text{MMM}\ldots\ldots \times 2^{-(2n+1)} & \text{for LSB } m_0 = 1 \\ (-1)^S \times 1.\text{MMM}\ldots\ldots \times 2^{-2n} & \text{for LSB } m_0 = 0 \end{cases}$$

Applying this interpretation we calculate the dynamic range of this system.

Maximum Value = $(-1)^S \times 1.1111111111111 \times 2^{-(2x0+1)} = (2 - 2^{-13}) \times 2^{-1}$

Minimum Value = $(-1)^S \times 1 \times 2^{-(2x14+1)} = 2^{-29}$

These maximum and minimum values can be verified from the graph as well.

Dynamic Range (in dB) = $20 \log_{10} \left[\frac{(2 - 2^{-13}).2^{-1}}{2^{-29}}\right] = 174.6$ dB

So we see that there is significant improvement in the dynamic range of the New Format as compared to fractional fixed point format and a greater SNR ratio as compared to various floating point formats. We have achieved our objective to derive an effective data format for 16-bit applications.

Chapter 6

Simulation and Performance Analysis

Simulation and Performance Analysis

We have arrived at a point where we finally consider different data formats for performance evaluation. Though these formats are not implemented on a real DSP processor but they have been simulated using Microsoft Visual C++ 2008 which provides at par evaluation of the data formats.

Now the main question which arises is that what would be the formats under consideration for simulation and comparison with the New Format. In the previous sections, we have seen the two existing solutions for 16-bit processors. One of them is the fixed point format and the other is the floating point format. Out of the various floating point formats, we choose the s10e5 for comparison because it possesses a sufficiently high dynamic range of 252.86 dB. So finally we narrow down on the three formats which have to be simulated.

Table 6-1: Formats under Consideration

Format	Representation	Characteristics
Fixed Point Format	s15e0	High accuracy but low dynamic range
Floating Point Format	s10e5	Low accuracy but high dynamic range
New Format	s15r	Optimized for decent performance w.r.t accuracy as well as dynamic range

The simulation of each of these formats is performed in three phases for testing its usability and effectiveness in major DSP applications. The three step process that we went through includes the following.

1. Packing and Unpacking of a Sine Wave
2. Implementation of the three formats on a digital FIR Filter
3. OFDM implementation

All the simulation results will be compared with IEEE 32-bit standard floating point format in order to show the effectiveness of the data formats with respect to a 32-bit format. The simulation results are plotted in a time domain graph. The y-axis contains the ratio of the output in the format under consideration to the output in IEEE 32-bit standard floating point format expressed in dB.

The formula used for the plot is

$$y(n) = 20 \log_{10} \left[\frac{\text{Output in 16-bit format under consideration}}{\text{Output in IEEE 32-bit Standard floating point format}} \right]$$

Where n represents the number of time samples

Ideally the value of y(n) should be 0. So larger the deviation in the plot greater the error introduced by the format. The plot with minimum deviation from baseline represents the best performance.

6.1 Simulation Details

Our simulation procedure involves

- Converting each input into the specified format
- Performing all the calculations in the specified format with a larger bit accumulator
- Storing the accumulator value back into the specified format
- Re-converting the output from the specified format back to readable form

A simulation block diagram that describes this procedure is shown

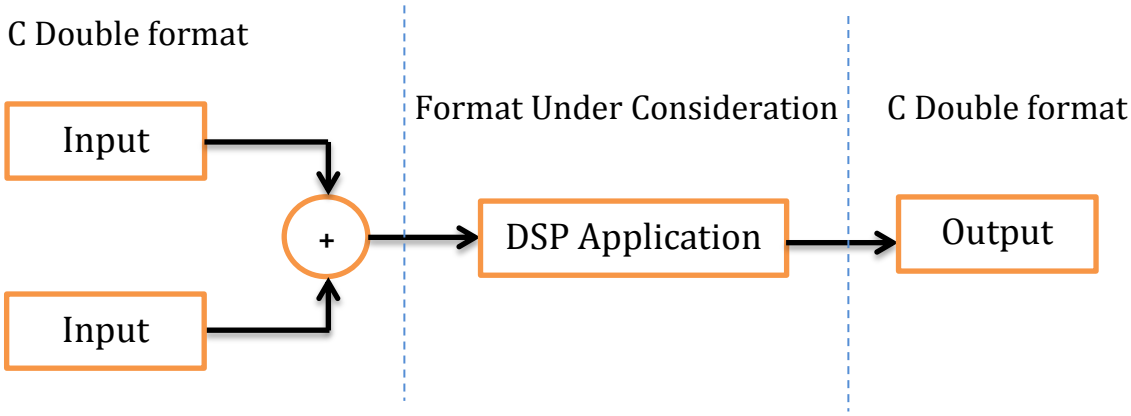

Figure 6-1: Simulation Block Diagram

The calculations performed for the DSP application in the specified format is achieved by operator overloading. Since most of the DSP applications use multiplication and addition operations for their implementation, so operators such as add (+), subtract (-), multiply (×), equal (=) and plus equal to (+=) are overloaded. Operator overloading performs operations on two classes or a single class and a constant. Each class contain a **val** variable representing the required format and a **vue** variable representing its equivalent double format. Performing arithmetic operations on the objects of the classes uses the val variable and performs the required calculation in the specific format. The result is stored back in the same format by the overloaded = operator. The accumulator is taken to be of double format providing a 32-bit accumulator for the DSP applications.

Table 6-2: Simulation Specifications

Parameter	Specification
Classes	s15e0, s10e5, s15r
Packing Function	setValue()
Unpacking Function	getValue()
Overloaded Operators	+, -, ×, =, +=, Negation
DSP Algorithms	FIR Filter, FFT, OFDM

6.2 Phase I - Packing and Unpacking of a Sine Wave

Packing and Unpacking a sine wave became the first step of our simulation for two main reasons.

- setValue( ) and getValue( ) functions had to be checked for storing and retrieving values to and from the specified format respectively.
- Operator overloading was avoided in this step to reduce complexity and check the format's integrity.

The following steps were performed for simulation

(a) Quantizing a sine wave from 0 to 2π into 1024 discrete values

(b) Packing each of these values into all the three formats by the functions s15e0::setValue( ), s10e5::setValue( ) and s15r::setValue( )

(c) Unpacking the results obtained in step (b) back to double format by the functions s15e0::getValue( ), s10e5::getValue( ) and s15r::getValue( )

(d) Comparing the original input with the output obtained after packing and unpacking by using the formula described earlier and plotting the curve

$$y(n) = 20 \log_{10} \left[\frac{\text{Output in 16-bit format under consideration}}{\text{Input in IEEE 32-bit Standard floating point format}} \right]$$

$$\text{Where } 0 \leq n \leq 1024$$

The Output to Input curves in time domain obtained after plotting the values of simulation are shown below.

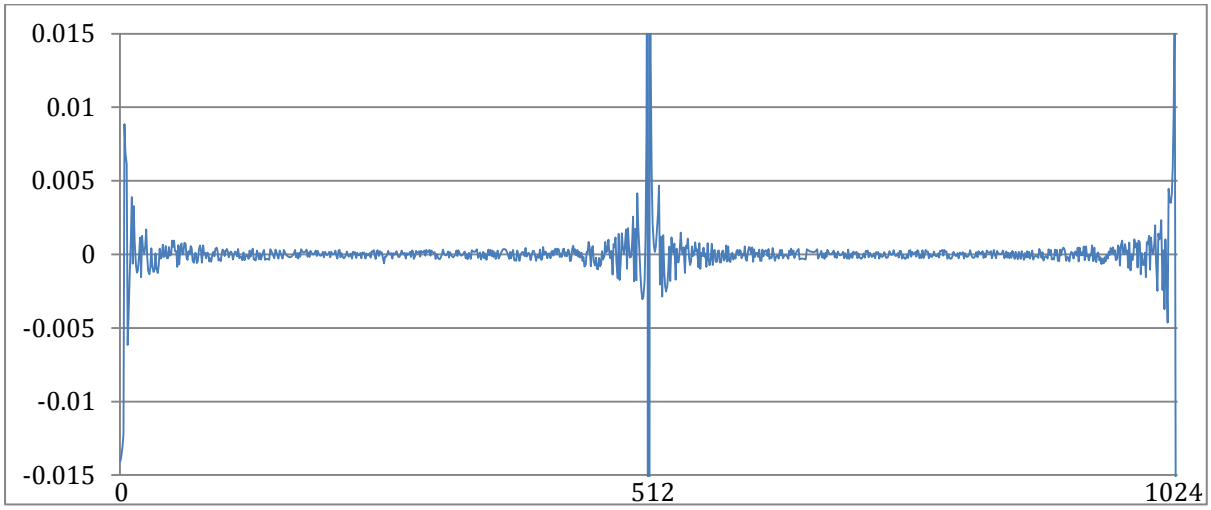

Figure 6-2: Fixed Point Format (s15e0)

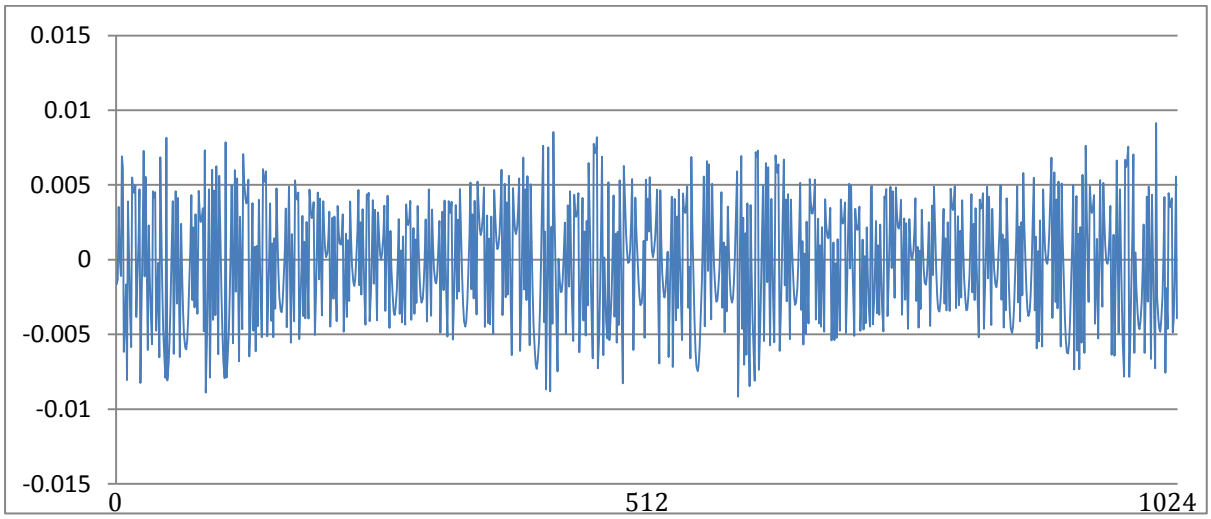

Figure 6-3: Floating Point Format (s10e5)

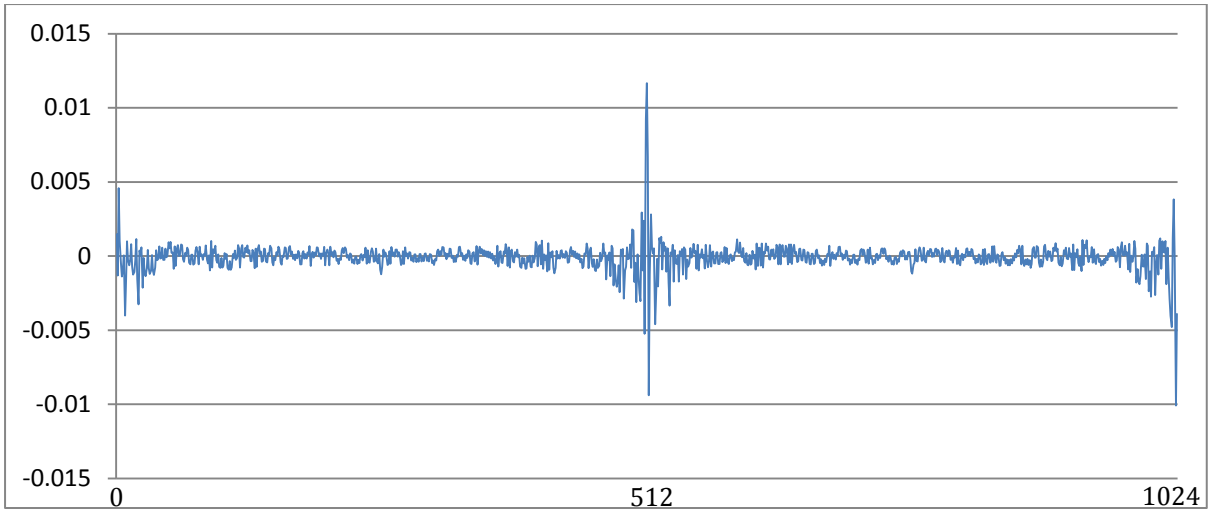

Figure 6-4: New Format (s15r)

From Figure 6-2, it can be seen that the fixed point format is excellent when it comes to signals of higher magnitude. However its incapability of storing signals of lower magnitude can be clearly seen as the deviation peaks up to as high as -0.21 dB at 512 (π) and 0.04 dB at 0 and 1024 (2π). From Figure 6-3, we can easily derive that the floating point format shows an average deviation of ±0.003 dB throughout the range and is not suitable for signals of higher magnitude. Figure 6-4 clearly shows the advantage of the New Format over the fixed and floating point format. For signals of larger magnitude it has comparable performance to fixed point format whereas for signals of lower magnitude the maximum deviation obtained is only 0.012 dB which is way better than the fixed point format. As expected the New Format performs better than the other two formats under consideration.

6.3 Phase II - Implementation of a digital FIR Filter

FIR filter was implemented mainly because of the following two reasons.

- The algorithm is very simple to implement.
- Only two basic operators (+) and (*) are overloaded. So it provides an insight into operator overloading before moving into complex DSP applications.

Table 6-3: FIR Filter Specifications

Parameter	Value
Sampling Frequency	2 Hz
Type	Low Pass
Cut-off Frequency	0.15 Hz
Order	68 (69 taps)
Gain(0 - 0.1 Hz)	1 (ripple = 0.05 dB)
Gain(0.2 – 1 Hz)	0 (attenuation = -70 dB)

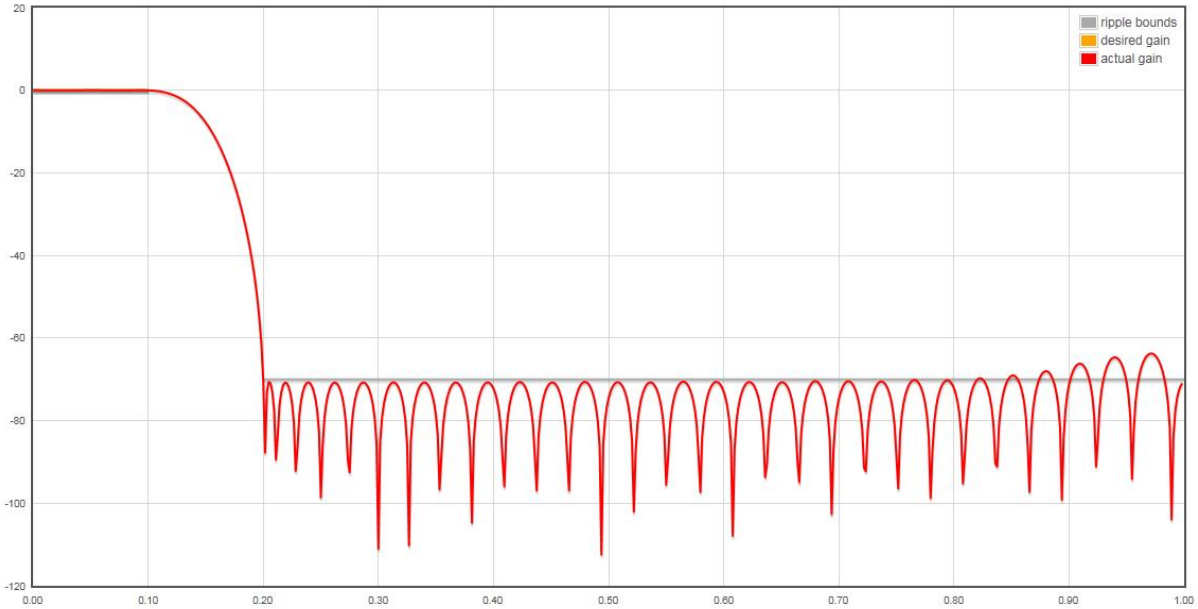

Figure 6-5: Frequency Response of the FIR Filter

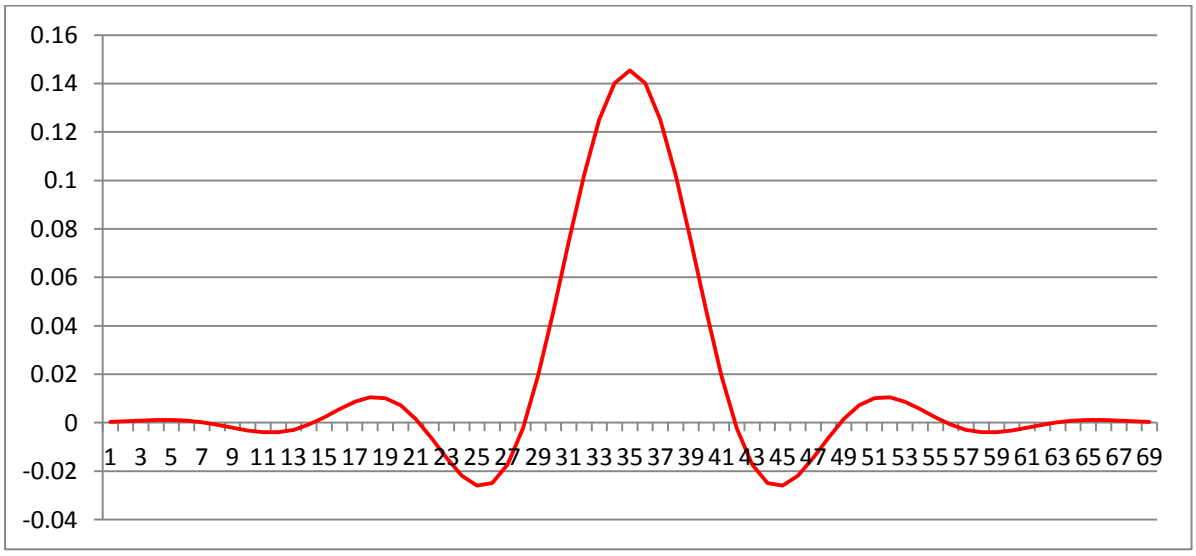

Figure 6-6: Impulse Response of the FIR Filter

The following steps were performed for simulation

(a) Generating of FIR filter coefficients as per the filter specifications mentioned above.

(b) Generation of a two toned sine wave with frequency f_1 below cut-off frequency and frequency f_2 above cut-off frequency.

(c) Passing the two toned signal through the digital low pass filter in all the three formats as well as standard 32-bit floating point format.

(d) Comparing output obtained from the 16-bit formats with respect to the output obtained from the 32-bit standard format and plotting the deviation curve.

$$y(n) = 20 \log_{10} \left[\frac{\text{Output in 16-bit format under consideration}}{\text{Output in IEEE 32-bit Standard floating point format}} \right]$$

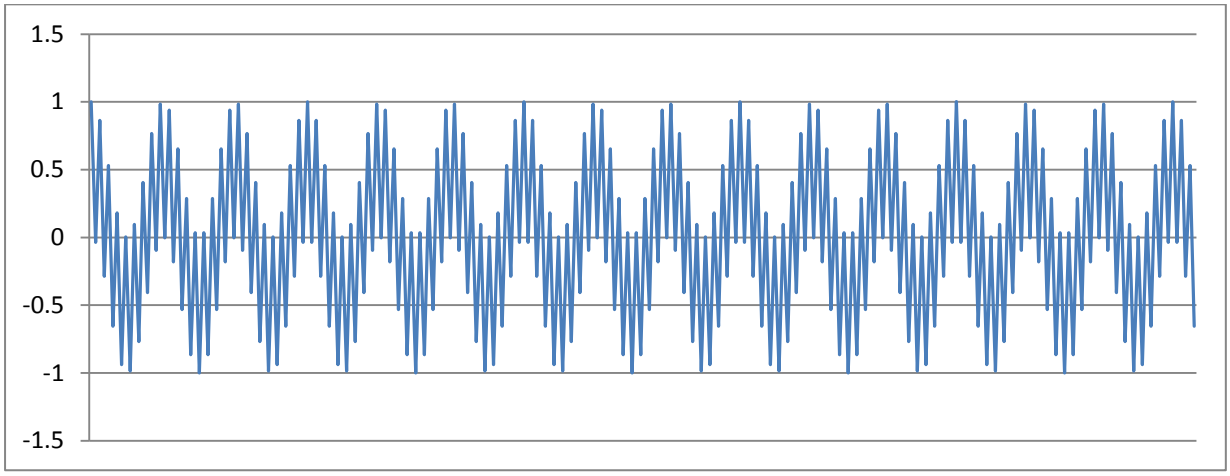

Figure 6-7: FIR Input – Two Toned Sine Wave

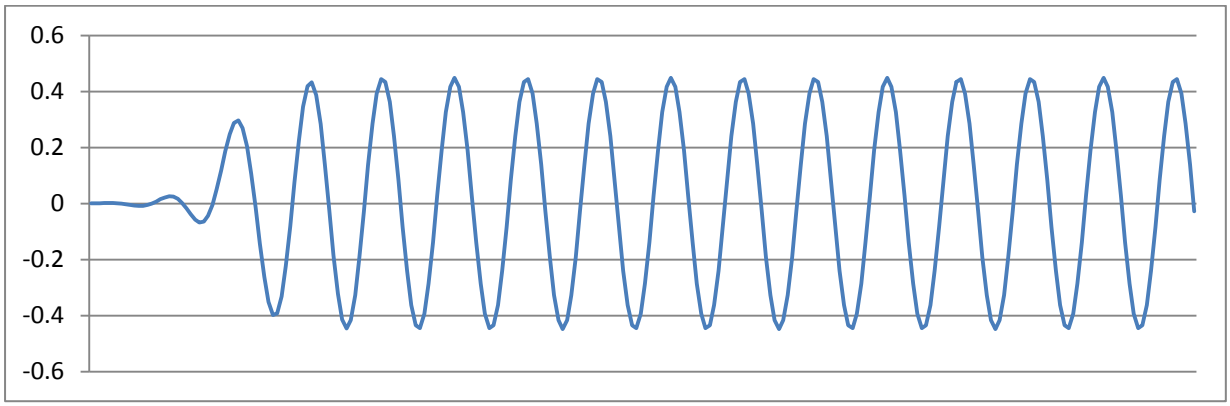

Figure 6-8: FIR Output – Low Frequency Signal

The Output to Input curves in time domain obtained after plotting the values of simulation are shown below.

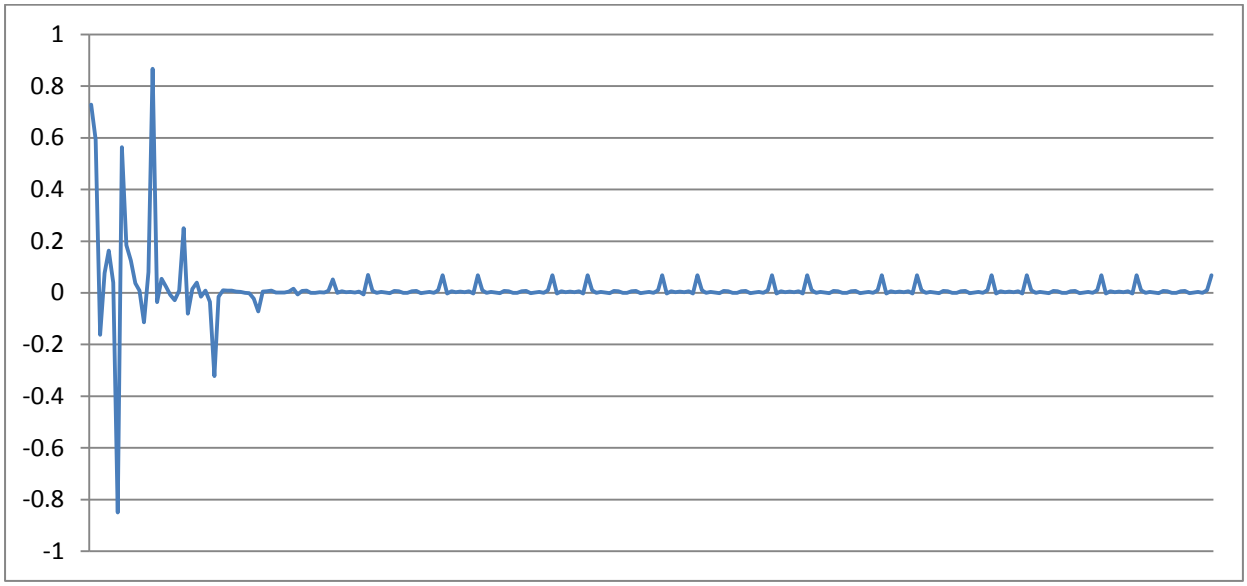

Figure 6-9: Fixed Point Format (s15e0)

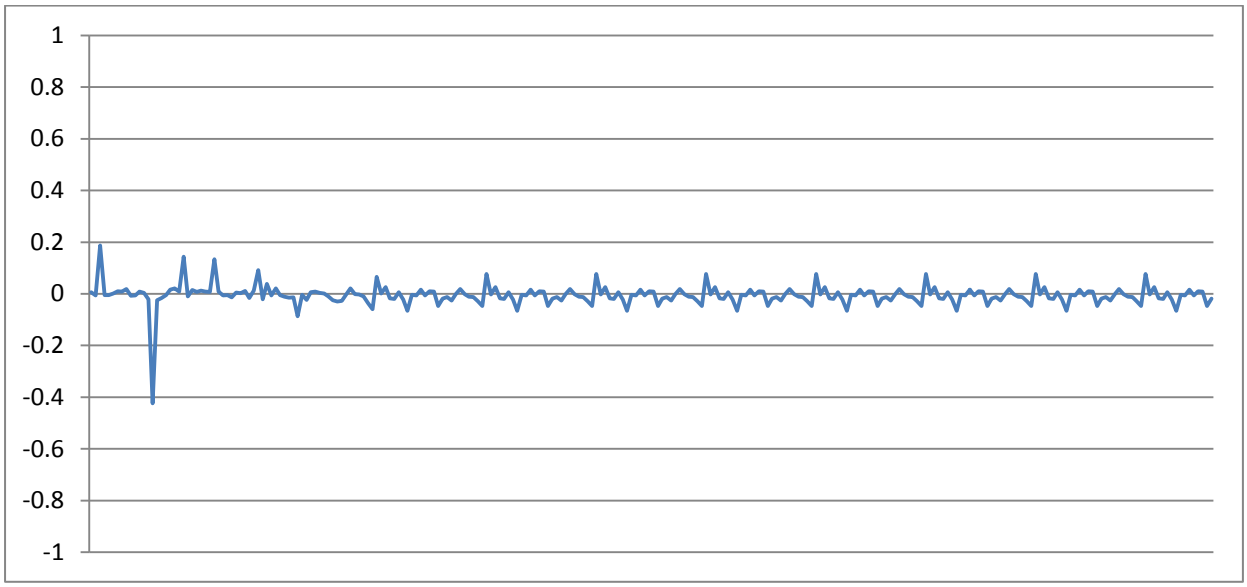

Figure 6-10: Floating Point Format (s10e5)

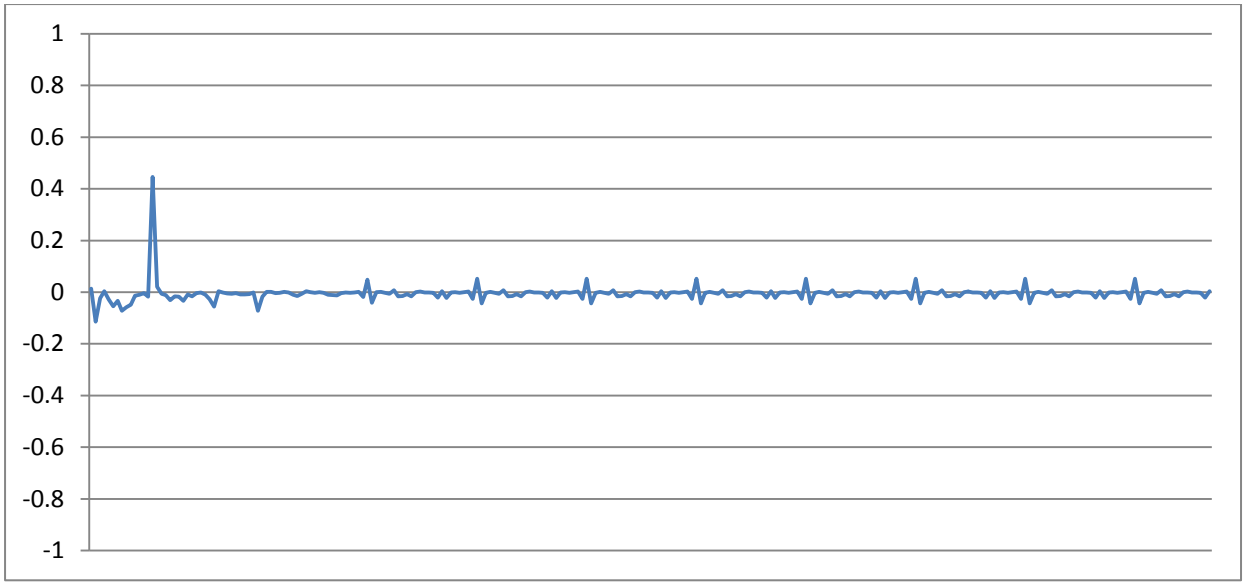

Figure 6-11: New Format (s15r)

Similar results were obtained in this simulation as in phase I. When the filter starts, due to lack of previous time samples it takes 69 samples for the filter to obtain the original low frequency output. During this time interval, the output signal is of low magnitude and hence fixed point formats show a deviation of as high as 0.85 dB. On the other hand floating point formats show deviations from baseline throughout the filter operation. The New Format as expected shows a deviation of 0.45 dB for lowest value and remains almost close to baseline throughout. Hence in this case as well we see that the New Format is the clear winner.

6.4 Phase III – OFDM Implementation

We reach at the final point of our project completion. The OFDM module has a set of transmitter and receiver. One of them possesses an IFFT block and the other has an FFT block. So the main region where the three formats are put to test is the FFT block which involves some heavy calculations.

The vector input to the FFT block is first packed into the specified format. All the internal variables are declared as C-double type to provide a larger accumulator. After performing all the calculations in the FFT block, the double type accumulator values are packed back into the specified format. The performance is biased by the FFT algorithm because it has different noise characteristics than other DSP algorithms. In order to remove the noise and

represent weaker signals, we use Automatic Gain Control (AGC) techniques while implementing the receiver and transmitter modules. The AGC algorithm is applied each time the data enters the FFT module for some vigorous calculation.

The details of OFDM theory, simulation model and algorithm are provided in the next chapter for a better understanding of the model. Finally the simulation results are summarized and a comparison between the three data formats is made to find out the best of them.

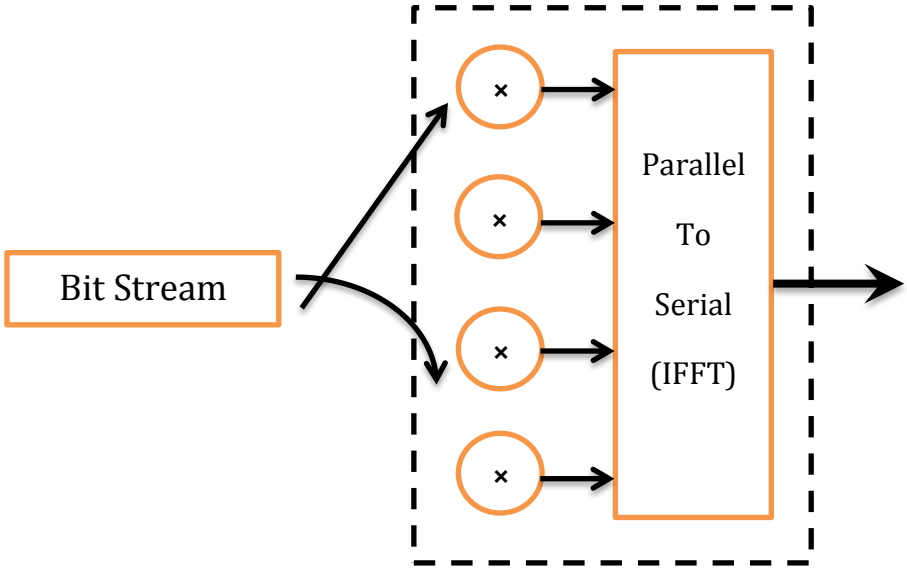

Figure 6-12: Functional Diagram of an OFDM Signal Creation

6.5 Summary

As far as Phase I and Phase II of our simulation is concerned it is clear that the New Format performs identical to fixed point format for signals of larger magnitude allowing minimal noise. However for signals of smaller magnitude the New Format completely leaves behind the fixed point format with very accurate representation. The floating point format is not even close to the New or fixed point format for signals of larger magnitude and hence should be avoided.

Chapter 7

Orthogonal Frequency Division Multiplexing and Its Implementation

Orthogonal Frequency Division Multiplexing and Its Implementation

Modulation – A mapping of the information on changes in the carrier phase, frequency, amplitude or combination of them.

Multiplexing – A method of sharing a common bandwidth with other independent data channels.

Orthogonal frequency division multiplexing (OFDM), essentially identical to coded OFDM (COFDM) and discrete multi-tone modulation (DMT), is a process of digital modulation which is configured to split a communication signal in several different channels. It is essentially a combination of **modulation** and **multiplexing**. A large number of closely-spaced orthogonal sub-carriers are used for carrying data. The data is divided into several parallel data streams or channels, one for each sub-carrier. Each of these channels is formatted into a narrow bandwidth modulation, with each channel operating at a different frequency. The process of OFDM makes it possible for multiple channels to operate within close frequency levels without impacting the integrity of any of the data transmitted in any one channel.

OFDM has developed into a popular scheme for wideband digital communication, whether wireless or over copper wires, used in applications such as digital television and audio broadcasting, wireless networking and broadband internet access.

Advantages:

- Makes efficient use of the spectrum by allowing overlap.
- By dividing the channel into narrowband flat fading sub channels, OFDM is more resistant to frequency selective fading than single carrier systems are.
- Eliminates ISI and IFI through use of a cyclic prefix.
- Using adequate channel coding and interleaving one can recover symbols lost due to the frequency selectivity of the channel.
- Channel equalization becomes simpler than by using adaptive equalization techniques with single carrier systems.

- It is possible to use maximum likelihood decoding with reasonable complexity.
- OFDM is computationally efficient by using FFT techniques to implement the modulation and demodulation functions.
- Is less sensitive to sample timing offsets than single carrier systems are.
- Provides good protection against co-channel interference and impulsive parasitic noise.

Disadvantages:

- The OFDM signal has a noise like amplitude with a very large dynamic range; therefore it requires RF power amplifiers with a high peak to average power ratio.
- It is more sensitive to carrier frequency offset and drift than single carrier systems due to leakage of the DFT.

7.1 Characteristics and principles of operation

7.1.1 Orthogonality

In OFDM, the name itself says orthogonal which means that the subcarriers used for data transmission are orthogonal to each other. Orthogonality ensures that there is no cross-talk between the subcarriers which would otherwise lead to data loss. Since there is no cross talk between subcarriers, inter carrier guard bands are not required which simplifies OFDM module designs.

Another major advantage of orthogonality is that it allows high spectral efficiency which implies that the frequency band available can be utilized to maximum.

But it has its disadvantage as well. OFDM requires high frequency synchronization between the receiver and transmitter. Without proper synchronization the subcarriers would lose their orthogonality. This results in inter carrier interference (ICI). Some major causes for frequency non-synchronization are

- Mismatched receiver and transmitter oscillator.
- Doppler shift (especially due to multipath).

7.1.2 Implementation using the FFT algorithm

The orthogonality allows for efficient modulator and demodulator implementation using the FFT algorithm on the receiver side, and inverse FFT on the sender side.

The IFFT output along with the cyclic prefix serves as the transmitted data whereas the FFT output on the receiver side gives the original data input to the transmitter.

Forward FFT takes a random signal, multiplies it successively by complex exponentials over the range of frequencies, sums each product and plots the results as a coefficient of that frequency. The coefficients are called a spectrum and represent "how much" of that frequency is present in the input signal. The results of the FFT in common understanding is a frequency domain signal.

We can write FFT in sinusoids as

$$X(k) = \sum_{n=0}^{N-1} \left(x(n) \sin \frac{2\pi kn}{N} - j\, x(n) \cos \frac{2\pi kn}{N} \right)$$

Here x(n) are the coefficients of the sines and cosines of frequency 2πk/N , where k is the index of the frequencies over the N frequencies, and n is the time index. x(k) is the value of the spectrum for the k_{th} frequency and x(n) is the value of the signal at time n.

The inverse FFT takes this spectrum and converts the whole thing back to time domain signal by again successively multiplying it by a range of sinusoids.

The equation for IFFT is

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} \left(X(k) \sin \frac{2\pi kn}{N} + j\, X(k) \cos \frac{2\pi kn}{N} \right)$$

The difference between FFT and Inverse FFT is the types of coefficients the sinusoids are taking, and the minus sign, and that's all. The coefficients by convention are defined as time domain samples x(n) for the FFT and X(k) frequency bin values for the IFFT. The two processes are a linear pair. Using both in sequence will give the original result back.

7.1.3 Guard interval for elimination of intersymbol interference

Low symbol rate modulation schemes are used in OFDM. The channel time characteristics are relatively small as compared to the symbol length. This results in less interference by multipath propagation which is otherwise high for high-

rate stream. Accounting for the larger duration of symbols, it is easier to insert guard bands into the symbols preventing intersymbol interference.

The cyclic prefix takes care of the guard interval. It is inserted at the beginning of each symbol by extracting a certain number of bits from the end of the symbol. The reason for this is that the receiver will integrate over an integer number of sinusoid cycles for each of the multipaths when it performs OFDM demodulation with the FFT.

We have made use of the above three characteristics in our simulation as that is all required for receiver and transmitter design. Some other characteristics of OFDM that we haven't used in simulation and are out of the scope of the thesis are

- Simplified Equalization
- Channel Coding and Interleaving
- Adaptive Transmission
- Space Diversity

7.2 Idealized system model

The ideal system model for implementing an OFDM process contains two modules – the transmitter and the receiver. Before the binary data enters the transmitter it is modulated as per specified modulation schemes such as BPSK, QPSK, and QAM etc. This modulated data now acts as the input to the transmitter.

The **transmitter** module converts the serial data into parallel data (i.e. sends the data via orthogonal sub-carriers). The parallel data is then sent into the IFFT block which performs an IFFT operation on it. The subsequent process involves adding of cyclic prefix to the IFFT output. Each set of IFFT output is appended to form a serial data which is finally transmitted via the channel.

The **receiver** on the other hand performs the exact inverse of the transmitter. The received serial stream of data is broken down into chunks of data from which the cyclic prefix is removed. The resultant output is the IFFT output at the transmitter. So this data is fed into a FFT block to obtain the desired parallel data (i.e. data from orthogonal sub-carrier). The parallel data is finally appended to obtain the original modulated signal. This modulated signal is then demodulated to obtain the original binary data.

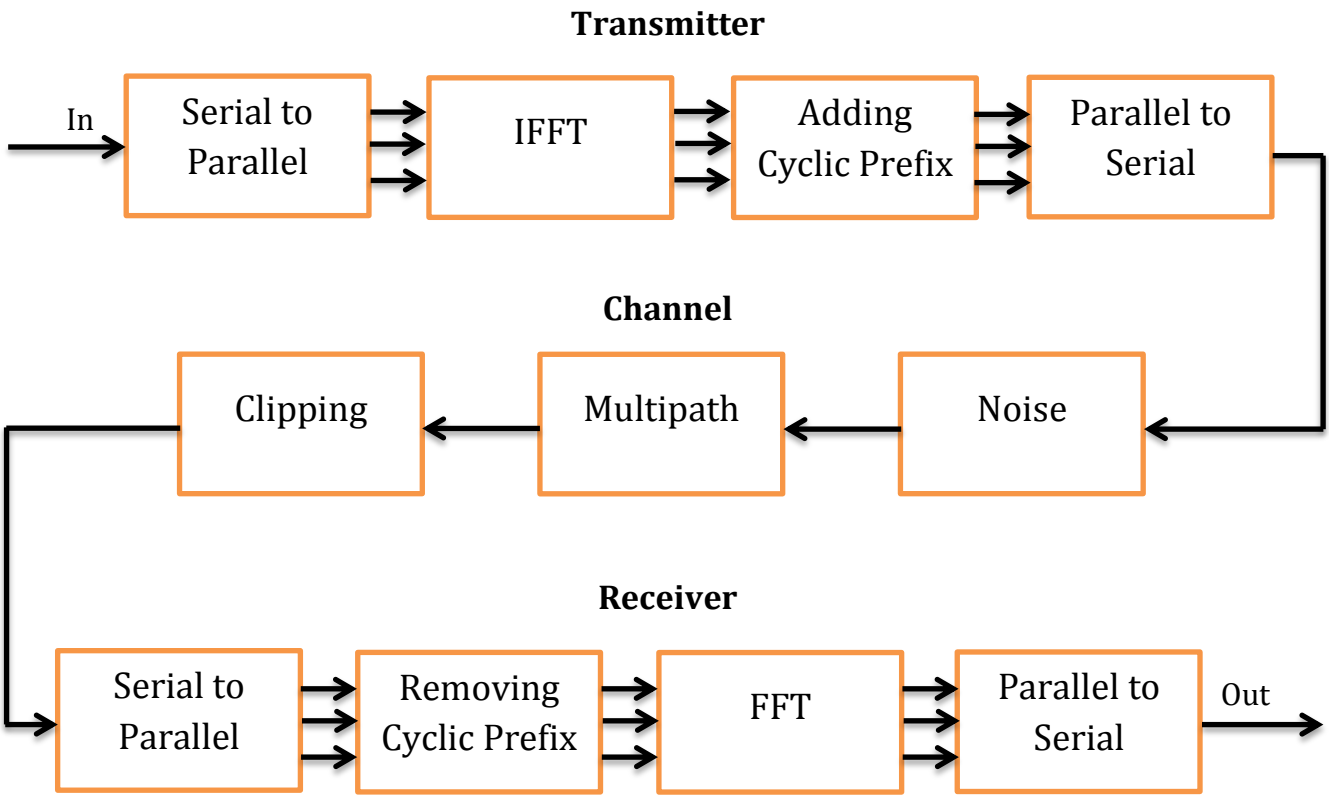

Figure 7-1: OFDM Simulation Flowchart

7.3 OFDM simulation as per IEEE 802.11a specification

For an effective and easy implementation of OFDM in wireless transmission, we chose a system loosely based on IEEE 802.11a specifications.

7.3.1 Simulation Parameters

The OFDM parameters used in IEEE 802.11a protocol is mentioned in the table below which serves as the basis of our simulation.

Table 7-1: IEEE 802.11a Parameters

Parameters	Value
FFT size, nFFT	64
Number of used subcarriers, nDSC	52
FFT Sampling frequency	20MHz
Subcarrier spacing	312.5kHz
Used subcarrier index	{-26 to -1, +1 to +26}
Cyclic prefix duration, T_{cp}	0.8us or 16 units
Data symbol duration, T_d	3.2us or 64 units
Total Symbol duration, T_s	4us or 80 units
Modulation Scheme	BPSK or 1 bit/sample

7.3.2 Simulation Model

The following steps were performed in the OFDM simulation model.

Modulation

(a) Generation of random binary sequence

(b) BPSK modulation i.e. bit 0 represented as -1 and bit 1 represented as +1

Transmitter

(c) Assigning input sequence to multiple OFDM symbols where data subcarriers from -26 to -1 and +1 to +26 are used (Serial to Parallel conversion). The subcarrier indexed 0 is filled with 0s corresponding to dc input

(d) Performing an Inverse FFT operation on each parallel set of data after appropriate rearrangements and 0 paddings

(e) Adding cyclic prefix of 16 bits from the end of the IFFT output at its beginning to give 80 bit output

(f) Appending the IFFT outputs to form a single serial stream of data to be transmitted

(g) Using Welch method of power calculation for signal transmitted

Receiver

(h) Grouping the received vector into multiple symbols and removing the cyclic prefix

(i) Performing an FFT calculation on each symbol to obtain the original data and then performing the appropriate rearrangement to obtain the desired sequence.

(i) Demodulation and conversion to bits

Error Calculation

(g) Subtracting the Input Sequence from the output sequence to calculate the number of error bits

(h) Calculating the signal strength and error percentage

7.3.3 Simulation Results

The FFT and Inverse FFT block in the receiver and transmitter was implemented using operator overloading for comparing each of the three formats – fixed, floating and new format. The internal variables, which act as accumulator in a DSP processor, were assigned 32 bits (float type) for simulating larger MAC units in 16 bit processors. The channel module in the simulation flowchart was deliberately left out as it didn't have any significant effect on performance evaluation of the data formats and would only add to noise. A simple Octave script where a BPSK modulated signal is transmitted on the 52 used subcarriers is used for generation of the OFDM signal. After performing the simulation, the following results were obtained.

Table 7-2: Simulation Results of OFDM implementation for different data formats

Parameters	s15e0 (Fixed Point)	s10e5 (Floating Point)	s15r (New Format)
Total Number of Bits	8000	8000	8000
Number of Valid Bits	5132	3096	5248
Number of Error Bits	2868	4904	2752
Error Percentage	35.85 %	61.30 %	34.40 %
Signal Validity	64.15 %	38.70 %	65.60 %

Power Spectral Density vs. Transmit Spectrum (Welch method)

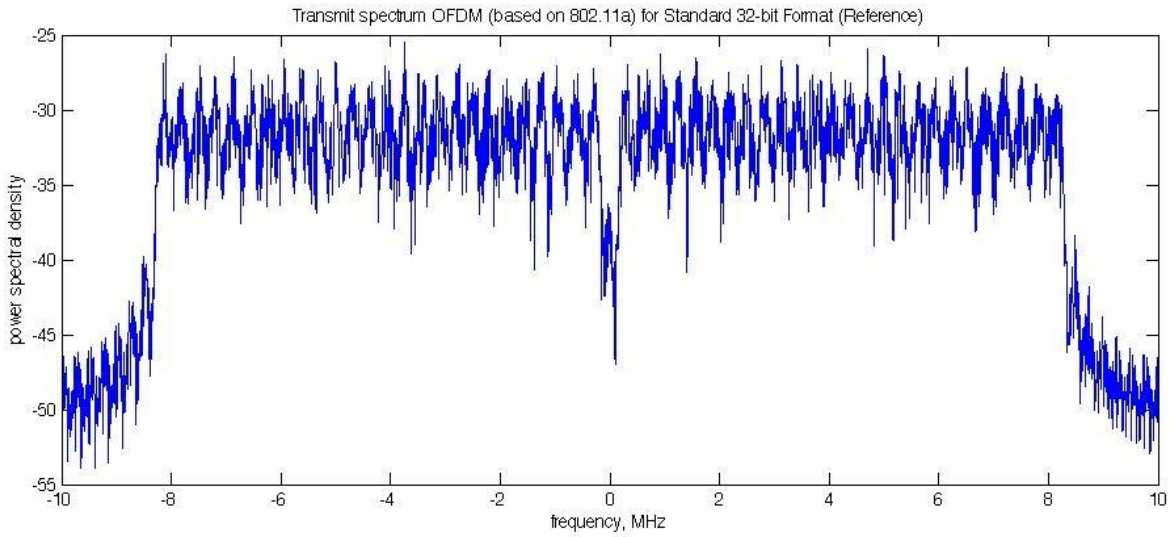

Figure 7-2: 32-bit Reference Format

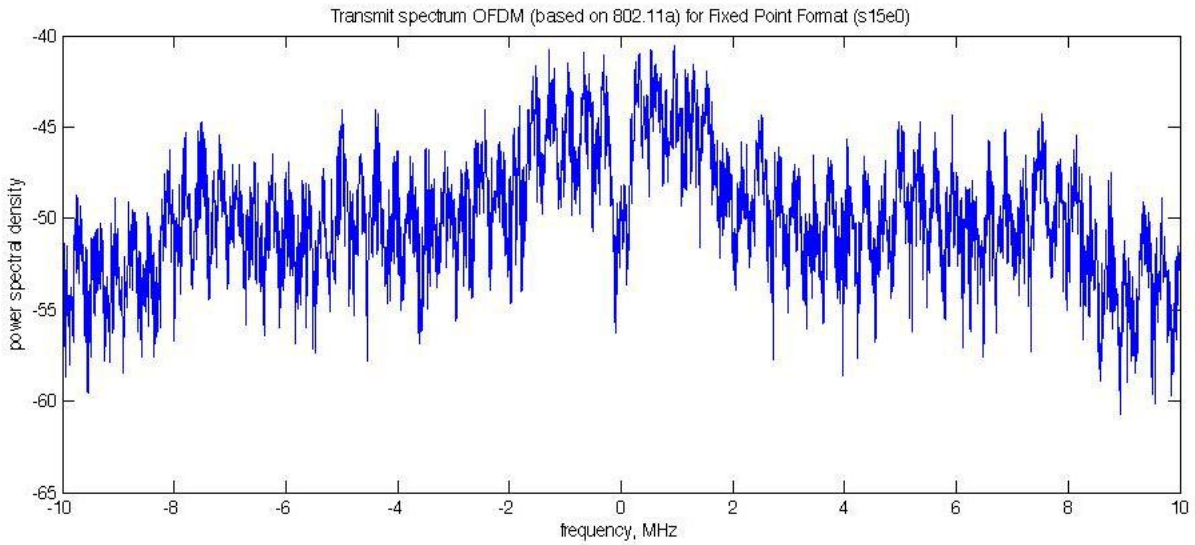

Figure 7-3: Fixed Point Format (s15e0)

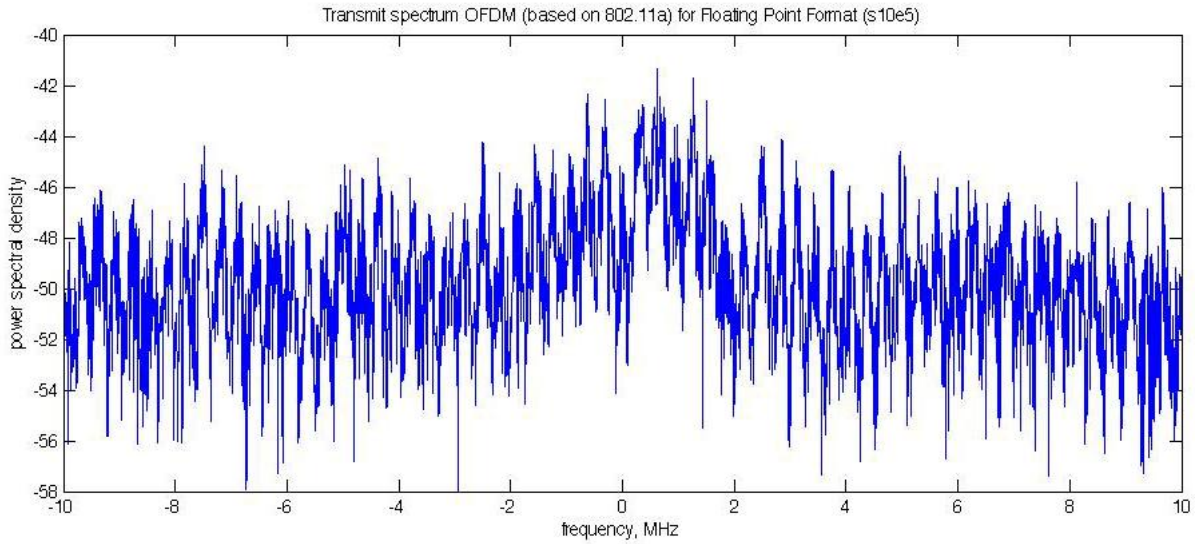

Figure 7-4: Floating Point Format (s10e5)

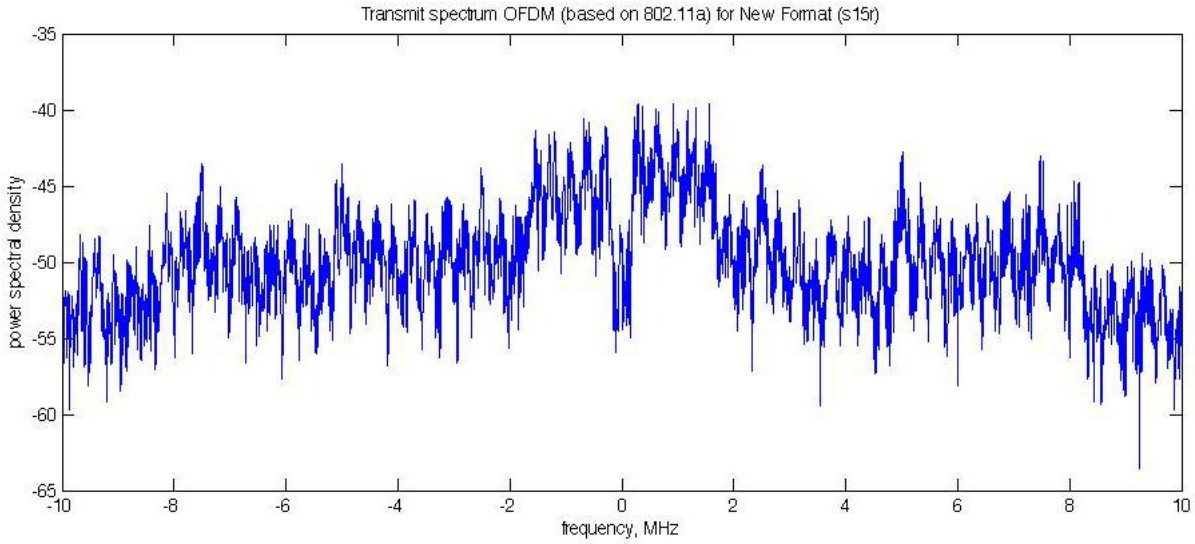

Figure 7-5: New Format (s15r)

From the table and the power spectral density plot, it is clearly seen that the New Format and fixed point format fairs well for an OFDM transmission. The floating point format is the worst of the lot. As expected there is a small advantage of the New Format over the fixed point format with a signal validity and accuracy of 65.60 %. The signal validity for the fixed point format just lacks behind the New Format with a meagre 1.45% difference. However the signal validity with the floating point format lags far behind the other two and is not at all suitable for being used in an OFDM transmission.

The power spectral density plot tells that higher the power spectral density greater is the strength and integrity of the signal at that frequency. The power spectral density plot for IEEE 32-bit Standard floating point format (Figure 7-2) clearly shows that leaving the zero frequency components at the start and end as well as at the centre (for the dc input 0) the complete frequency range transmits the signal with very high power spectral density. Comparing that with the New Format (s15r) and fixed point format (s15e0), signals between frequencies range -8 MHz to 8 MHz has an average power spectral density above -50. But the floating point format (s10e5) performs miserably having power spectral density well below -50 for each and every frequency component in the plot indicating very poor performance.

All in all we can finally conclude that 16-bit New Format is a good data format to be used in 16-bit processors which has some advantages over the fixed point format, though small in this case. The floating point format due to its lower SNR ratio is highly unsuitable for implementing an OFDM algorithm.

Chapter 8

Conclusions

Conclusions

In this chapter, we wrap up our investigation on new data formats for 16-bit DSP applications and discuss potential application for its implementation and scope of future work. We finally summarize the results obtained throughout the course of this thesis.

8.1 Potential Applications

Some typical and well-known items which contain embedded 16-bit DSPs:

- Cell phones
- Fax machines
- DVD players and other home audio equipment
- Cars (for example: the anti-lock braking system)
- Computer disk drives
- "Switch" at telephone company (more than a lot)
- Digital radios
- High-resolution printers
- Digital cameras

8.2 Scope of Future Work

The new format developed shows a significant improvement over the existing data formats. The format had a path into 16-bit high level language programming as proved by simulation. However a lot deeper study could be done to achieve better results than what we have obtained in this thesis.

1. From the class of floating point formats we could try deriving a number of different irregular data formats that could have a significant performance gain over the formats discussed in this thesis.

2. Hardware implementation of the formats could have been done which would provide a real world analysis for a more accurate study.

3. A closer study of OFDM could be made to include the channel into simulation so that an accurate analysis of the complete OFDM procedure could have been made.

4. We could examine different modules for OFDM and new data formats could have been developed which would be module specific and would provide a significant performance gain.

5. A complete set of C++ class could be developed for new data formats which would overload arithmetic, logic as well as relational operators.

6. The new class of floating point formats could have been expanded for 24-bits as well as 8-bits DSP applications.

7. Investigation could be undertaken to see if the new formats have appeal outside of the realm of digital signal processing.

8.3 Summary

In this thesis, we have gone through the shortcomings of 16-bit DSP applications. Two major problems of dynamic range and noise performance have been discussed. The problem of implementing the DSP applications on a High Level Language platform has been discussed. Various attempts were made to achieve significant improvement over existing data formats. An extensive simulation and comparison between the data formats was made and results were summarized for analysis. Superior performance in terms of increased dynamic range and reduced round-off noise has been demonstrated for the new formats over existing 16-bit formats. These new formats can improve the efficiency of developing both very small and very inexpensive DSP applications.

References

[1] M. Richey. “The Application of Irregular Data Formats for Improved Performance in 16-bit Digital Signal Processing Systems.” Master’s thesis, Univ. of Kansas, Dec. 2006.

[2] Steven W. Smith. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. [On-line].
Available: http://www.dspguide.com/pdfbook.htm [Oct 3, 2010].

[3] Manuel Richey. “A New Class of Floating-Point Data Formats with Applications to 16-Bit Digital-Signal Processing Systems.” IEEE Communications Magazine. [On-line]. 47(7), pp. 94 – 101.
Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5183478 [Aug 6, 2010].

[4] Gene Frantz & Ray Simar. “Comparing Fixed- and Floating-Point DSPs.”
Internet: http://focus.ti.com/lit/wp/spry061/spry061.pdf [Nov 2, 2010].

[5] Prof. W. Kahan. “Status of IEEE Standard 754 for Binary Floating-Point Arithmetic.”
Internet: http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF [Oct 25, 2010].

[6] M. Richey. “Re: Help regarding the new floating point number system in regard to 16-bit floating point format.” Personal e-mail [Nov 1, 2010].

[7] M. Richey. “Re: Dynamic Range Calculation.” Personal e-mail [Nov 10, 2010].

[8] Vilmos Pálfi and István Kollár. “Round off Errors in Fixed-Point FFT.”
Internet: http://mycite.omikk.bme.hu/doc/74336.pdf [Nov 11, 2010].

[9] “Operator Overloading.”
Internet: http://www.keithschwarz.com/cs106l/winter20072008/handouts/200_Operator_Overloading.pdf, Mar 03, 2008 [Nov 15, 2010].

[10] Sripriya R. “Operator Overloading (Part I).”
Internet: http://www.exforsys.com/tutorials/c-plus-plus/operator-overloading-part-i.html, Nov 12, 2010 [Dec 2, 2010].

[11] Sripriya R. “Operator Overloading (Part II).”
Internet: http://www.exforsys.com/tutorials/c-plus-plus/operator-overloading-part-ii.html, Feb 09, 2011 [Dec 2, 2010].

[12] Wagner, Brian and Michael Barr. "*Introduction to Digital Filters*," Embedded Systems Programming, December 2002, pp. 47-48. [Jan 10, 2011].

[13] João Martins. “How to implement the FFT algorithm.” Internet: http://www.codeproject.com/KB/recipes/howtofft.aspx, Feb 02, 2005. [Jan 25, 2011].

[14] Vlodymyr Myrnyy. “A Simple and Efficient FFT Implementation in C++, Part I.” Internet: http://www.eetimes.com/design/signal-processing-dsp/4017495/A-Simple-and-Efficient-FFT-Implementation-in-C--Part-I, May 10, 2007 [Jan 25, 2011].

[15] Vlodymyr Myrnyy. “A Simple and Efficient FFT Implementation in C++, Part II.” Internet: http://www.eetimes.com/design/signal-processing-dsp/4017496/A-Simple-and-Efficient-FFT-Implementation-in-C--Part-II, May 25, 2007 [Jan 25, 2011].

[16] “Orthogonal Frequency Division Multiplex (OFDM) Tutorial.” Internet: http://www.complextoreal.com/chapters/ofdm2.pdf, [Mar 05, 2011].

[17] Alan C. Brooks and Stephen J. Hoelzer. “Design and Implementation of Orthogonal Frequency Division Multiplexing (OFDM) Signalling.” Internet: http://cegt201.bradley.edu/projects/proj2001/ofdmabsh/ OFDM_ Design_Proposal.pdf, Dec 08, 2000 [Mar 05, 2011].

[18] Krishna Shankar M. “Understanding OFDM Transmission.” Internet: http://www.dsplog.com/2008/02/03/understanding-an-ofdm-transmission/, Feb 03, 2008 [Mar 22, 2011].

[19] Krishna Shankar M. “Spectrum of an OFDM Transmit.” Internet: http://www.mathworks.com/matlabcentral/fileexchange/ 19189-spectrum-of-an-ofdm-transmit, Mar 13, 2008 [Mar 22, 2011].

[20] IEEE Computer Society. “IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements.” Internet: http://standards.ieee.org/getieee802/download/802.11-2007.pdf, Jun 12, 2007 [Apr 12, 2011].

[21] Wikipedia. “Orthogonal frequency-division multiplexing”. Internet: http://en.wikipedia.org/wiki/Orthogonal_frequency-division_multiplexing, May 01, 2011 [06 May 2011].